

# A Tool for Drawing Undirected Graphs

*B. W. Bush*

Energy and Environmental Analysis Group

Los Alamos National Laboratory

9 April 1996

I. Introduction .....	2
II. Theory .....	2
III. Software .....	4
A. TGraphLayout Class .....	4
B. Command Line Tool .....	7
C. Graphical Tool .....	7
IV. Usage .....	9
A. Strategies .....	9
B. Performance .....	11
C. Sample Results .....	12
V. Availability .....	14
VI. Acknowledgements .....	14
VII. References .....	14
VIII. Appendix: Help Files .....	15
A. Command Line Tool .....	15
B. Graphical Tool .....	16
IX. Appendix: Source Code .....	19
A. Graph Layout Class .....	19
B. Command Line Tool .....	27
C. Graphical Tool .....	30

---

\* Mailing address: TSA-4, MS F604, Los Alamos National Laboratory, Los Alamos, NM 87545 /  
bwb@lanl.gov

## I. Introduction

The problem of laying out, or *drawing*, a graph arises in a wide variety of contexts. Automatically drawing computer network (e.g., LAN or WAN) configurations, object-oriented class diagrams, or database entity-relationship diagrams are examples of graph drawing. Estimating the layout of street networks containing some intersections with unknown locations is an example of the problem of drawing a graph. This paper discusses an algorithm for drawing general undirected graphs that relies on constructing a dynamical system analogous to the graph and evolving the state of the system to an equilibrium configuration. This configuration provides an aesthetically pleasing layout of the graph. We present an implementation of the algorithm as a C++ class. We also demonstrate the use of the class in a command-line executable program compilable on a variety of computer platforms as well as in an interactive 32-bit Windows program that animates the layout process.

## II. Theory

The algorithm presented here is a force-directed technique for graph drawing similar to that introduced by Eades [1,2] in 1984. The method involves representing the vertices of the graph with masses and representing the edges of the graph with springs. Optionally, additional springs may be added between those vertices not connected by an edge. The graph can then be treated as a dynamical system and allowed to converge to an equilibrium configuration. If the spring constants are appropriately chosen, the forces on the system lead to an equilibrium that has an aesthetically pleasing layout. See references [2] and [3] for a discussion of other force-directed techniques and for a general overview of graph drawing and visualization.

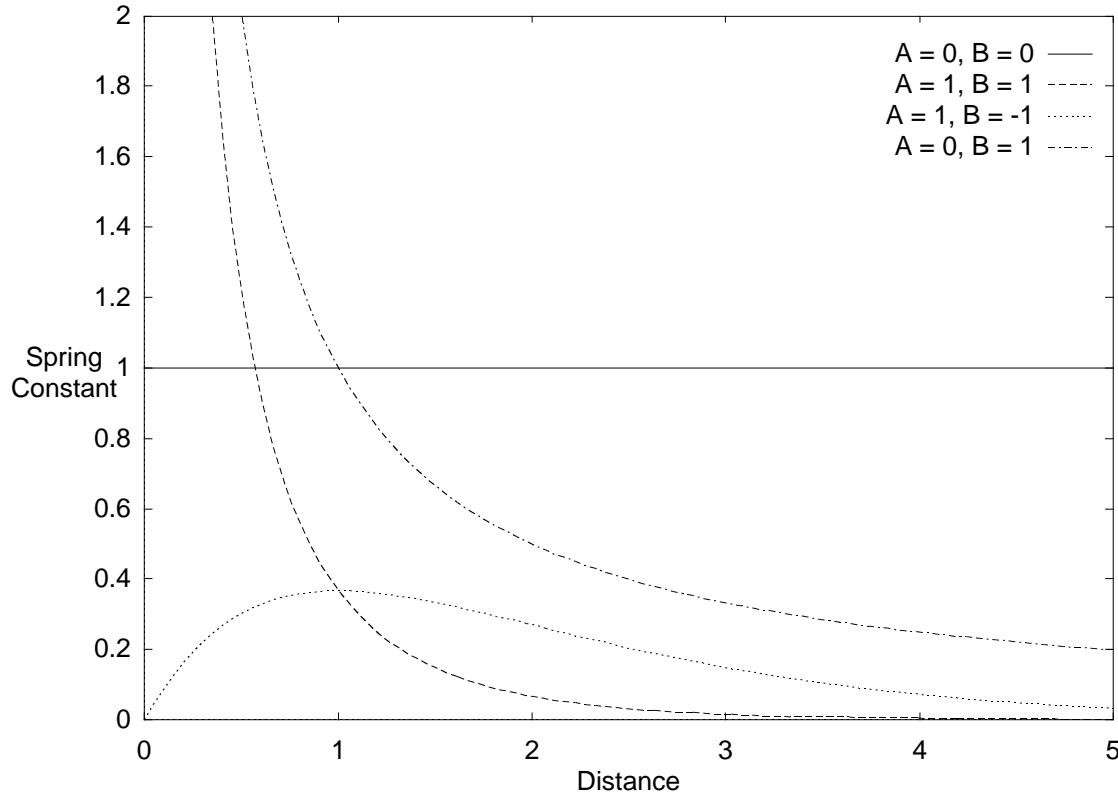
The graph's equation of motion is essentially that for the Brownian motion of particles coupled by harmonic oscillators. We start with unit masses representing the vertices which have positions given by two-dimensional vectors  $\mathbf{x}_i$ . The spring between vertices  $i$  and  $j$ , with equilibrium length  $\ell_{ij}$ , has a spring constant  $k_{ij}$ . In addition to the force from the spring, we include a damping force and a random (thermal) force in the equation of motion:

$$\frac{d^2\mathbf{x}_i}{dt^2} + \gamma k_i \frac{d\mathbf{x}_i}{dt} + \sum_{j \neq i} k_{ij} (D_{ij} - \ell_{ij}) \hat{\mathbf{D}}_{ij} = \mathbf{R}_i(t) ,$$

where  $k_{ij} \neq 0$  only if there is a spring between vertices  $i$  and  $j$ . Here the magnitude of this displacement is written  $D_{ij} = |\mathbf{D}_{ij}|$  and the direction is written  $\hat{\mathbf{D}}_{ij} = \mathbf{D}_{ij} / D_{ij}$ , where  $\mathbf{D}_{ij} = \mathbf{x}_i - \mathbf{x}_j$  is the displacement vector from vertex  $j$  to vertex  $i$ . Reference [4] provides a detailed overview of the coupled harmonic oscillator problem in mathematical physics. We use a damping factor  $\gamma k_i$  with  $k_i = \sqrt{\sum_{j \neq i} k_{ij}^2}$  so that the motion is critically damped for  $\gamma \approx 1$ . For the random force we use independent Gaussian random vectors  $\mathbf{R}_i(t)$  such that  $\langle \mathbf{R}_i(t) \rangle = \mathbf{0}$  and  $\langle \mathbf{R}_i(t) \mathbf{R}_j(t') \rangle = 2T\gamma k_i \delta_{ij} \delta(t-t') \mathbf{1}$ , where  $T$  is

the temperature of the system. For a discussion of the *fluctuation-dissipation* theorem, which relates the damping  $\gamma$  to the temperature  $T$ , see reference [5].

To allow flexibility in the choice of how the spring constant depends upon the length of the spring, we have adopted the spring constant equation  $k_{ij} \equiv e^{-A\ell_{ij}} / \ell_{ij}^B$ , where  $A$  and  $B$  are constants. The spring constant  $k$  as a function of length  $\ell$  for several values of  $A$  and  $B$  is shown in Figure 1.



**Figure 1. Dependence of the spring constant  $k(\ell) \equiv e^{-A\ell} / \ell^B$  on the length of the spring  $\ell$  for several values of the constants  $A$  and  $B$ .**

The greater the value of  $k(\ell)$ , the more constrained springs with length  $\ell$  are. Thus values of  $A$  and  $B$  which have appreciable  $k$  only for small  $\ell$  will produce graph layouts where the local structure of the graph is emphasized at the expense of the global structure; values of  $A$  and  $B$  which have appreciable  $k$  for large  $\ell$  will produce graph layouts where the global structure is emphasized at the expense of the local structure. By “tuning” the values of  $A$  and  $B$ , one can optimize the layout for a particular type of graph. It is necessary to have some flexibility here because different types of graphs have different degrees of connectivity and other properties. For example, a computer network layout looks different from an object-oriented class diagram.

As mentioned above, one can optionally connect all vertices to each other with springs—not just the vertices connected by edges. These additional springs are added to the system, and we set their lengths to the graph-theoretic distance between the vertices at

their ends: i.e., the length of the spring between two vertices is equal to the length of the shortest path between the vertices. We used an adjacency matrix representation of the graph to calculate the distances between all pairs of vertices efficiently. Let  $L_{ij}^{(1)}$  be the distance between vertices  $i$  and  $j$ , defined by

$$L_{ij}^{(1)} = \begin{cases} 0 & i = j \\ \ell_{ij} & i \text{ and } j \text{ are connected by an edge} \\ \infty & \text{otherwise} \end{cases}.$$

Thus the shortest distance between vertices  $i$  and  $j$ , traveling through at most  $n$  edges is [6]

$$L_{ij}^{(n+1)} = \min_k \{ L_{ik}^{(n)}, L_{kj}^{(n)} \}, \quad n \geq 1.$$

For large enough  $n$ , the quantity  $L_{ij}^{(n)}$  converges to a constant matrix. This constant matrix yields the shortest-path distances between all pairs of vertices in the graph.

Sometimes there is no natural length for the edges in a graph. In this case, it is useful to set the length of an edge to an optimal value for the layout. Although the best choice for this length depends on the specific type of graph, one can make a reasonable guess at edge lengths based on the degree of connectivity (i.e., the number of edges) at each of the edge's vertices. When a vertex has many edges connected to it, these edges should be long enough that their other vertices are not in each other's way—especially if those vertices also have many edges connected to them. Thus we intuit that making the edge length a function of the degree of connectivity at the edge's vertices will provide a semi-optimal separation of the vertices in the graph. The software discussed below supports several edge-length functions.

### III. Software

#### A. *TGraphLayout Class*

The *TGraphLayout* class implements functions to alter and optimize the layout of a graph. It conforms to the draft ANSI C++ standard and uses only standard C++ library and template library functions. The *TGraphLayout* class is designed to be self-contained, light weight, and portable. Because of these design considerations, we have chosen not to implement separate graph, vertex, edge, vector, and matrix classes, but rather to centralize the layout functions in a single class—the aim here is not to develop a class library for graph theory.

The class provides several member functions for setting up a layout:

```
// Construct an instance.  
TGraphLayout();  
  
// Set the number of vertices.  Return whether the  
// operation succeeded.  
bool SetVertexCount(short count = 0);  
  
// Set the position of the specified vertex to (x, y),  
// making it fixed in the layout if indicated.  Return  
// whether the operation succeeded.  
bool SetVertexPosition(short vertex, double x, double y,  
                      bool fixed = false);  
  
// Connect the specified vertices with an edge of the  
// specified length.  Return whether the operation  
// succeeded.  
bool ConnectVertices(short fromVertex, short toVertex,  
                     double length = 1);  
  
// Get the number of vertices.  Return whether the  
// operation succeeded.  
bool GetVertexCount(short& count) const;  
  
// Get the position of the specified vertex and whether it  
// is fixed in the layout.  Return whether the operation  
// succeeded.  
bool GetVertexPosition(short vertex, double& x, double& y,  
                      bool& fixed) const;  
  
// Get the velocity of the specified vertex.  Return  
// whether the operation succeeded.  
bool GetVertexVelocity(short vertex, double& vx, double&  
                      vy) const;
```

It is also possible to randomize the positions and velocities of the graph vertices. This can be useful for getting the graph layout out of a local minimum.

```
// Randomize the positions of the vertices up to a maximum  
// of the bound.  Return whether the operation succeeded.  
bool RandomizePositions(double bound = 1);  
  
// Randomize the velocities of the vertices up to a maximum  
// of the bound.  Return whether the operation succeeded.  
bool RandomizeVelocities(double bound = 1);
```

If the graph edges have no natural length or if conforming to the natural length is not a goal for the layout, edge lengths can be set automatically based on a pre-defined calculation scheme:

```
// There are several ways to calculate the edge length from
// the count of edges at either end of the edge:
enum EEdgeLengthStyle {
    kUni,           // unity
    kMin,           // the minimum
    kSqrtMin,       // the square root of the minimum
    kMax,           // the maximum
    kSqrtMax,       // the square root of the maximum
    kSum,           // the sum
    kSqrtSum,       // the square root of the sum
    kSqrtSumSqr,   // the square root of the sum of the
                    // squares
    kHarmSum,       // the harmonic sum
    kSqrtHarmSum   // the square root of the harmonic sum
};

// Set the lengths of the edges based on the specified edge
// length style. Return whether the operation succeeded.
bool SetEdgeLengths(EEdgeLengthStyle style = kSqrtMin);
```

The TGraphLayout class offers control over all parameters in the equation of motion for the graph:

```
// Set the time step for the calculation to the specified
// value. Return whether the operation succeeded.
bool SetTimeStep(double timeStep = 0.1);

// Set the temperature for the calculation to the specified
// value. Return whether the operation succeeded.
bool SetTemperature(double temperature = 0);

// Set the damping factor for the calculation to the
// specified value. Return whether the operation
// succeeded.
bool SetDamping(double damping = 1);

// Set the spring parameters for the calculation to the
// specified value. Return whether the operation
// succeeded.
bool SetSpringParameters(double a = 0, double b = 1);

// Get the time step for the calculation.
void GetTimeStep(double& timeStep) const;

// Get the temperature for the calculation.
void GetTemperature(double& temperature) const;

// Get the damping factor for the calculation.
void GetDamping(double& damping) const;

// Get the spring parameters for the calculation.
void GetSpringParameters(double& a, double& b) const;
```

Finally, several functions exist for laying out the graph:

```
// Initialize the calculation, connecting all of the
// vertices with springs if indicated.  Return whether the
// operation succeeded.
bool Initialize(bool connectAll = true);

// Return whether the calculation has been initialized.
bool IsInitialized() const;

// Evolve the vertex positions the specified number of time
// steps.  Return whether the operation succeeded.
bool EvolvePositions(long steps = 1);

// Get the kinetic and potential energy of the layout.
// Return whether the operation succeeded.
bool GetEnergy(double& kinetic, double& potential) const;

// Lay out the vertices automatically using the specified
// absolute and relative tolerances for the kinetic energy
// per vertex.  Return whether the absolute tolerance was
// met.
bool LayoutAutomatically(double epsilonAbsolute = 1e-4,
                        double epsilonRelative = 1e-8);
```

Note that the `TGraphLayout::EvolvePositions` function provides a low-accuracy solution to the equation of motion sufficient for the purpose of laying out a graph—a high-accuracy solution would slow the calculation unnecessarily.

Internally, the `TGraphLayout` class uses a nested class `TGraphLayout::TPoint` for storing and manipulating vertex positions and velocities. The adjacency matrix and spring length matrix for the graph are stored as Standard Template Library vector template class instances. An initialization flag is maintained to keep track of whether or not the spring length matrix has been calculated. For a graph with many vertices where all of the vertices are connected with springs, this calculation can take a long time.

## B. *Command Line Tool*

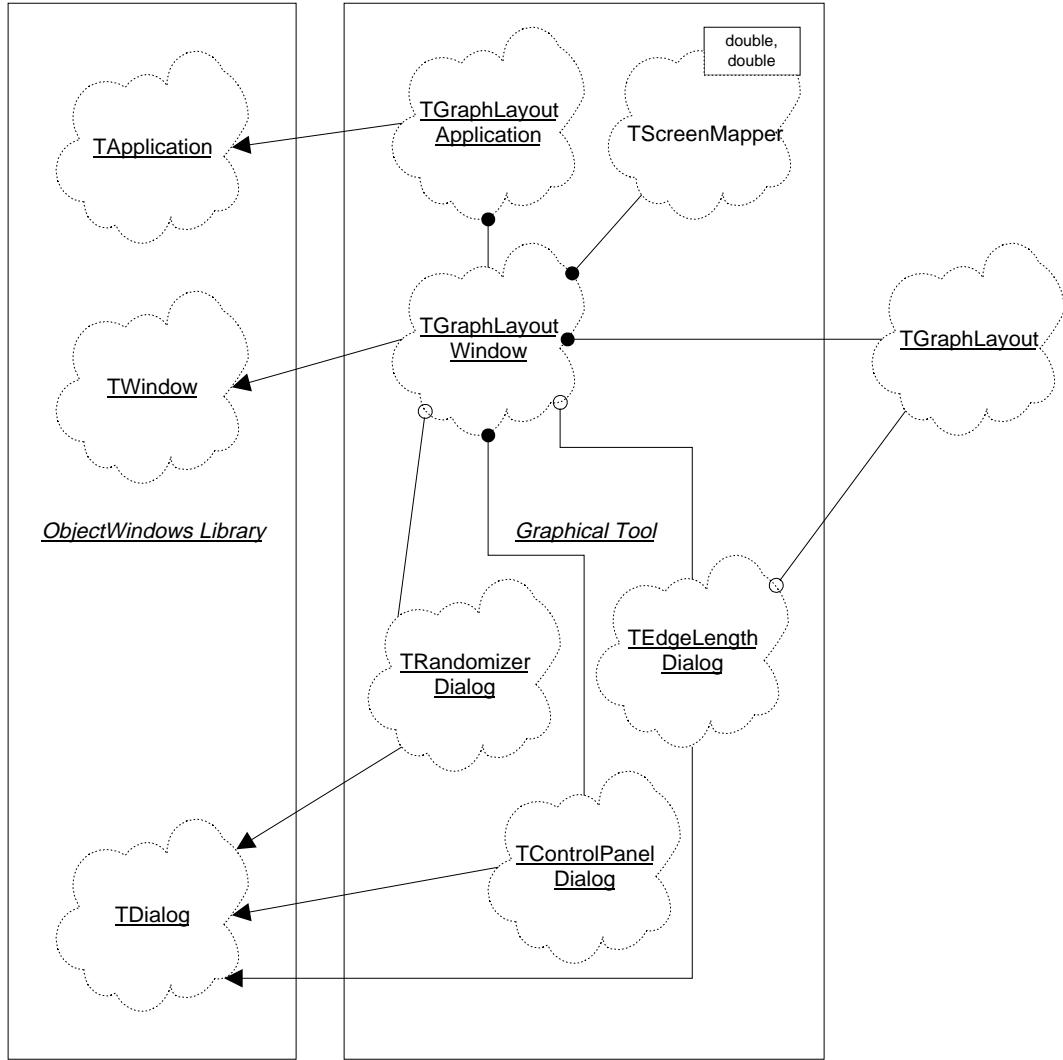
We provide a command line tool as a simple interface to the `TGraphLayout` class. The tool's source code conforms to the draft ANSI C++ standard and uses only standard C++ library and template library functions. Thus it is portable to any platform with a ANSI-compliant C++ compiler and standard libraries. The appendices list the help screen and the source code for the tool.

The tools features access to most of the `TGraphLayout` member functions. Only the randomization features and choice of edge length calculation are not supported.

## C. *Graphical Tool*

We also provide a graphical tool as alternative interface to the `TGraphLayout` class on 32-bit Windows platforms (Microsoft Windows NT and Windows 95). The tool's source code conforms to the draft ANSI C++ standard and uses standard C++

library and template library functions as well as Borland International's ObjectWindows Library (OWL). A simplified class diagram (in Booch notation) for the application is shown below:



**Figure 2.** A simplified Booch diagram for the graphical graph layout tool.

The `TGraphLayoutWindow` class handles most of the user interaction and passes information to its `TGraphLayout` instance; separate classes (`TRandomizerDialog`, `TEdgeLengthDialog`, and `TControlPanelDialog`) handle dialog-box interaction. Animation is achieved through the use of separate processor threads for the calculation and the user interface. The appendices list the help screen and the source code for the tool. Each class has a separate header and source file.

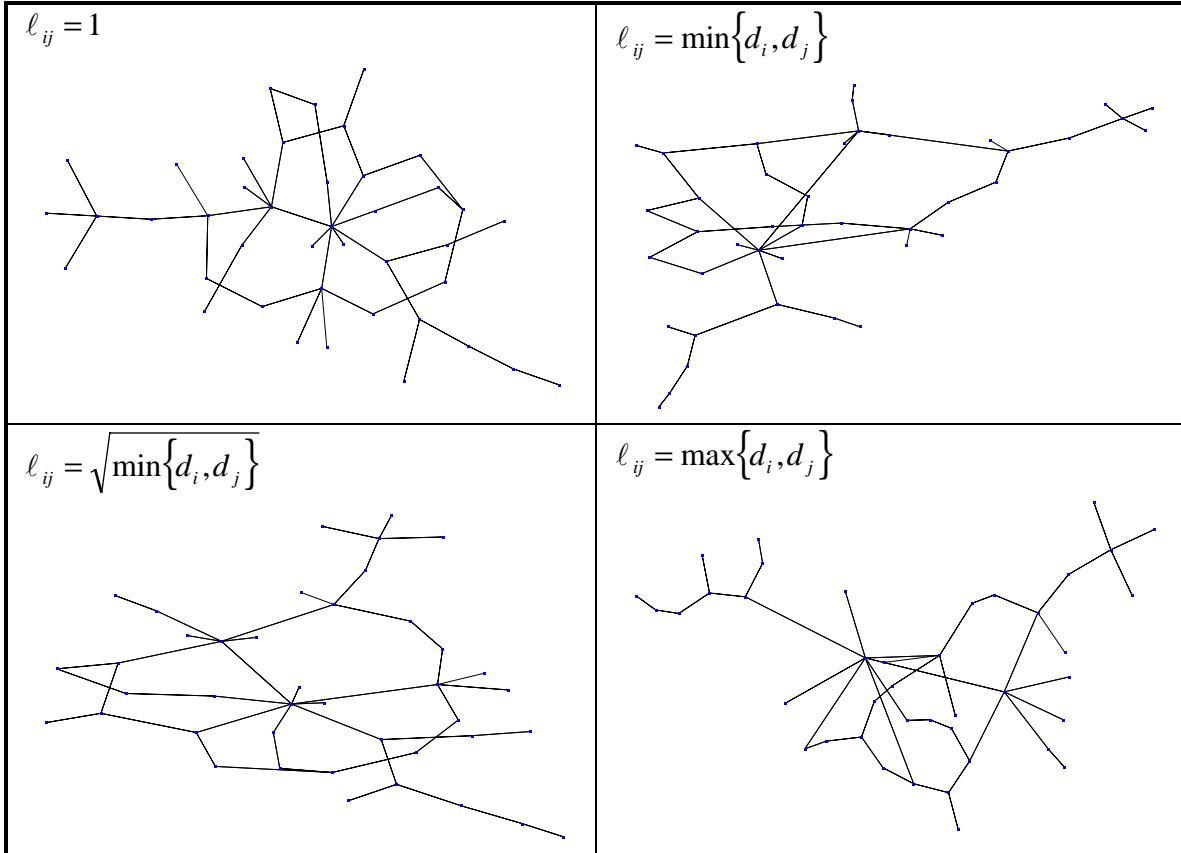
In addition to allowing interactive access to all of the `TGraphLayout` features, the graphical tool will also animate the solution of the graph's equation of motion. This provides a user with valuable visual feedback about the influence of the various

parameters (temperature, damping, and spring constants) upon the evolution of the graph layout. The user can also move vertices in the drawing and fix the positions of vertices to prevent them from moving.

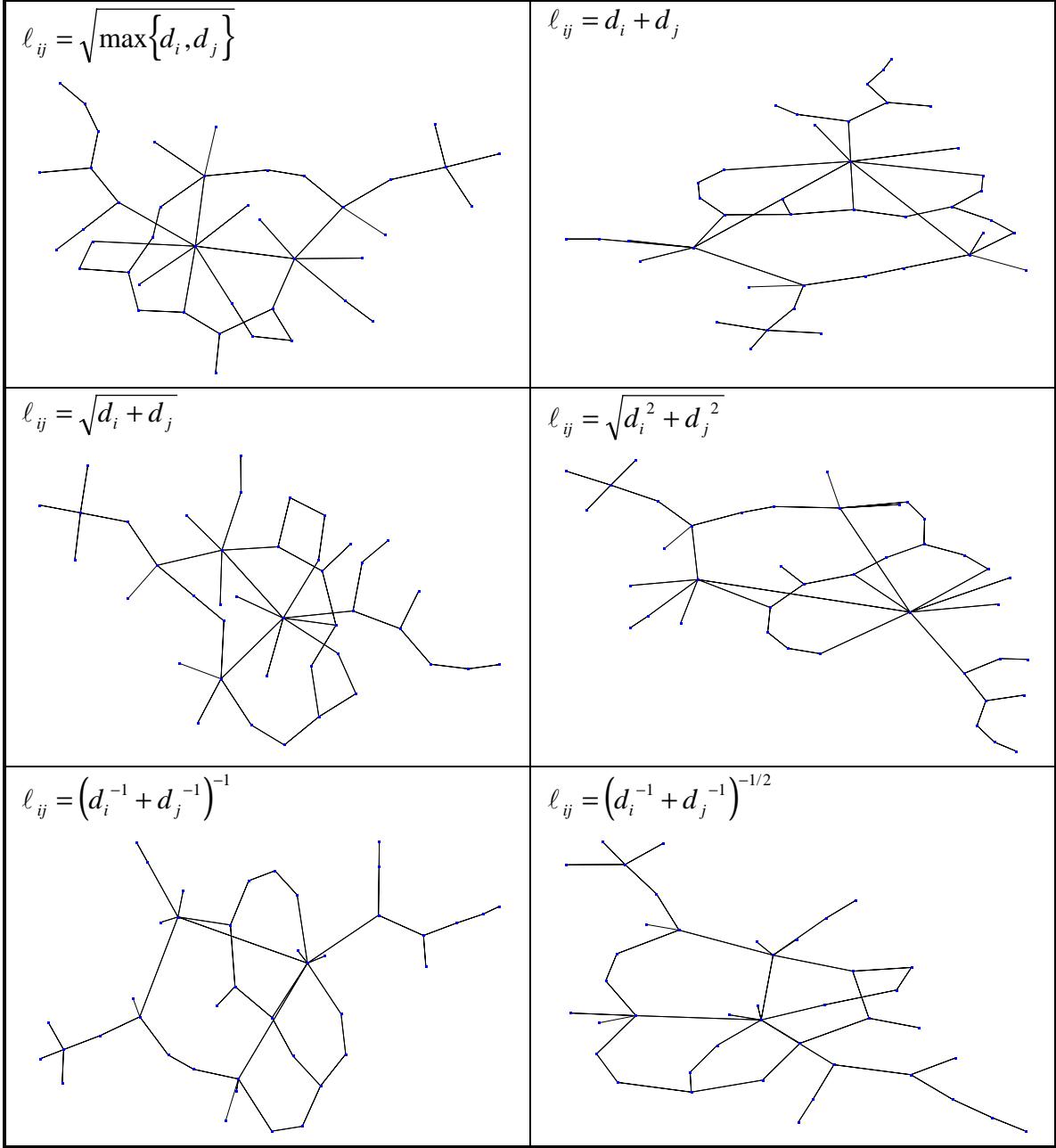
## IV. Usage

### A. Strategies

The graph layout tool will solve a variety of graph drawing problems. The simplest case is where only the connectivity of the graph is known. In this case we are free to set the edge lengths to whatever we like. (In the case where the edges have natural lengths that are given, it is probably desirable to retain those lengths as the spring lengths.) As discussed previously, an edge length useful for drawing the graph can be calculated from the degree of connectivity of the edge's vertices. We write the degree of connectivity at vertex  $i$  as  $d_i$ . The tool provides several edge length functions, shown below with example layouts based on them:



**Figure 3.** The edge length functions provided in the `TGraphLayout` class and a sample graph drawn with each function using the tool's automatic layout feature.



**Figure 3 (continued).**

The square root of the minimum,  $\ell_{ij} = (\min\{d_i, d_j\})^{1/2}$ , the square root of the sum,  $\ell_{ij} = (d_i + d_j)^{1/2}$ , and the square root of the harmonic sum,  $\ell_{ij} = (d_i^{-1} + d_j^{-1})^{-1/2}$ , typically work best. These three functions seem to set the edge lengths appropriately because they generate long edges only when both vertices have a high degree of connectivity, but avoid generating excessively long edges.

Usually it is best to connect all of the vertices with springs, not just those vertices with edges between them. If all vertices are not connected with springs, then there are no

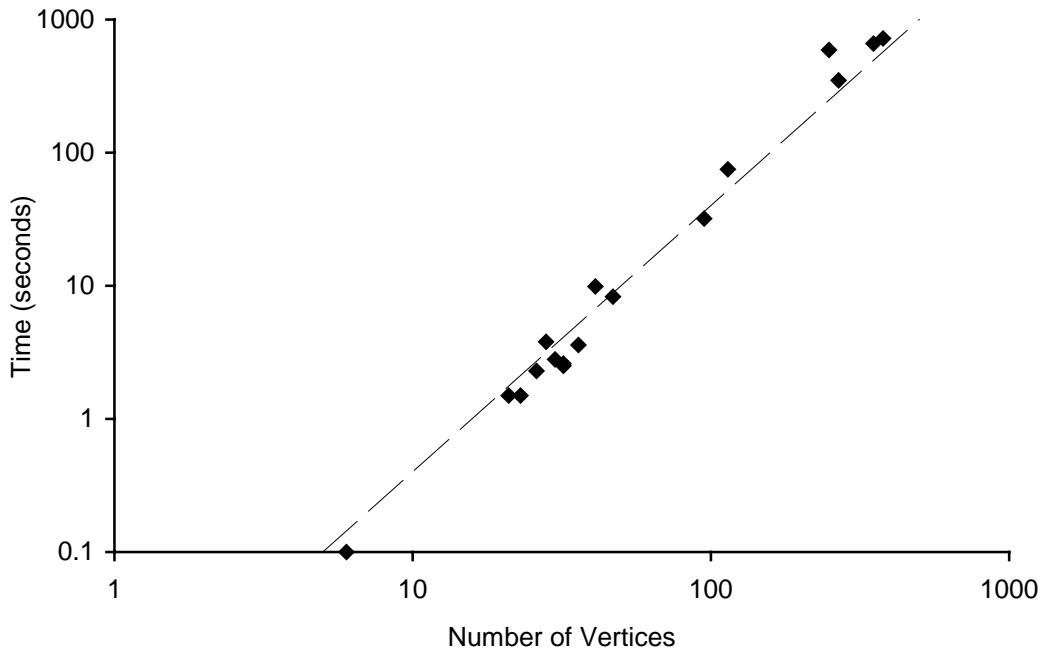
explicit constraints forcing vertices that are distant graph-theoretically to be far apart in the layout. The only situation in which it is useful not to connect all vertices with springs is when one wants the lengths of edges in the layout to be as close to the lengths of the edges in the graph as possible. If additional springs existed in this case, the springs along the edges would be compressed or expanded more than necessary due to the influence of the extraneous springs. Suppose, for example, that one wants to lay out a street network and that the lengths of the streets are known, but not the locations of the intersections; the tool can lay out a geographic representation of the street network. It would not make sense to add the “extra” springs in this case.

The `TGraphLayout` feature where a vertex’s position can be fixed so that it does not move, allows one to add additional constraints to the layout and incorporate known vertex locations in the layout. Also, by fixing some of the graph’s vertices, one can affect the layout.

The time step parameter determines the value of  $dt$  in the equation of motion; the smaller the time step, the more accurate the solution to the equation of motion. Since the accuracy of equation of motion’s solution is not a concern for drawing graphs, it is desirable to make  $dt$  fairly large, but not so large that the evolution becomes unstable due to numerical inaccuracy. The temperature parameter  $T$  determines how much thermal motion there is in the system. Thermal motion can be used to prevent the system from getting trapped in a configuration corresponding to a local minimum of its energy. The damping parameter  $\gamma$  regulates the rate at which the system loses energy and its configuration converges to a stable arrangement. If  $\gamma$  is too large, the system may converge too quickly, getting stuck in a local minimum of its energy.

## B. Performance

The graph above shows the performance of the algorithm on a 90 MHz Pentium computer running Windows NT.

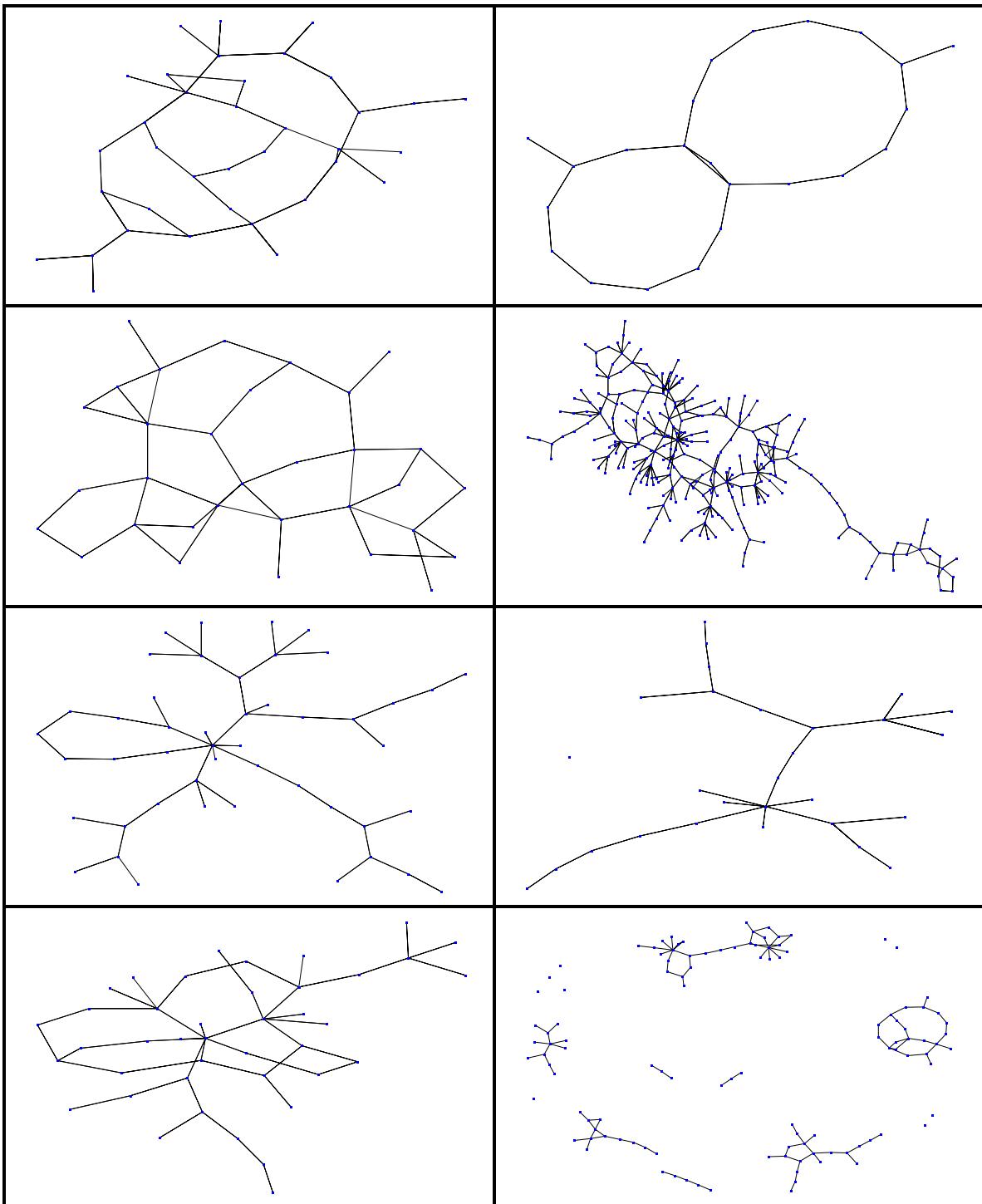


**Figure 4. Measured speed of the tool's automatic layout feature for sample graphs as a function of the number of vertices.**

The graph indicates that the computation time scales as the square of the number of vertices. This is expected theoretically.

### C. Sample Results

The diagrams below show some results of drawing several graphs using the tool's default settings and automatic layout feature.



**Figure 5.** Sample graph drawings using the tool's automatic layout feature.

The tool generally manages to separate both the large-scale and small-scale structure of the graph; however, unnecessary small-scale overlaps occur in some cases.

## **V. Availability**

All of the source code, executable programs, and examples discussed in this paper are available via anonymous FTP at <ftp://bwb.lanl.gov/pub/GraphLayout.tar.Z>.

## **VI. Acknowledgements**

This work was supported by the U.S. Department of Energy. Alan Berscheid assisted testing the graphical tool and Michelle Silva proofread the manuscript.

## **VII. References**

1. P. Eades, "A Heuristic for Graph Drawing," *Congressus Numerantium*, **42** (1984) 149, as cited in [2] below.
2. I. F. Cruz and R. Tamassia, *How to Visualize a Graph: Specification and Algorithms*, (available at <ftp://ftp.cs.brown.edu/pub/papers/compgeo/>, 1994).
3. G. Di Battista, P. Eades, R. Tamassia, and I. G. Tollis, "Algorithms for Drawing Graphs: An Annotated Bibliography," *Computational Geometry: Theory and Applications*, **4** (1994) 235, section 4.1.
4. A. L. Fetter and J. D. Walecka, *Theoretical Mechanics of Particles and Continua*, (McGraw-Hill Book Company: New York, 1980), chapter 4.
5. N. G. van Kampen, *Stochastic Processes in Physics and Chemistry*, (North-Holland: New York, 1981, section VIII.8.
6. A. Gibbons, *Algorithmic Graph Theory*, (Cambridge University Press: New York, 1985), section 1.3.1.

## VIII. Appendix: Help Files

### A. Command Line Tool

#### Usage:

```
GraphLayoutCL [-h] -v filename -e filename -o filename [-i number]
               [-s number] [-t number] [-d number] [-k number number]
               [-a] [-A]
```

#### Parameters:

```
-h    display help
-v    vertex filename
-e    edge filename
-o    output filename
-i    number of iterations [default = 1000]
-s    time step [default = 0.01]
-t    temperature [default = 0]
-d    damping [default = 1]
-k    spring parameters [default = 0 1]
-a    connect all vertices with springs [default = no]
-x    ignore lengths in edge file and use optimized edge lengths instead
-A    proceed automatically until convergence (parameters istdk ignored)
```

#### File formats:

The Vertex File must be a tab-delimited text file ending with a blank line. Each line represents a vertex. The fields it contains are x-position, y-position, and a flag (0 or 1) indicating whether the vertex is fixed. The Edge File must be a tab-delimited text file ending with a blank line. Each line represents an edge. The fields it contains are the index of the first vertex (counting from zero) in the vertex file, the index of the second vertex (counting from zero), and the distance between the vertices.

#### Algorithm:

The Layout Algorithm solves a mechanical model where each edge is represented by a spring. (Optionally, all vertices may be connected to one another with springs of length equal to the shortest-path distance between the respective vertices.) The springs have a spring constant given by the equation  $k = \exp(- A d) / d^B$  (where  $d$  is the distance between the vertices) and are damped. The vertices are also subject to thermal random forces (as in Brownian motion).

## B. Graphical Tool

---

### Graph Layout Help

**Graph Layout** is a tool for laying out the vertices of a graph. It uses a mechanical model based on placing springs between vertices as a layout algorithm.

#### **Menu commands**

##### *File* menu

- Open* reads a formatted vertex and a formatted edge data file.
- Close* clears the data.
- Save* saves the vertex data.
- Save as* saves the vertex data under a different name.
- Exit* exits the application.

##### *Edit* menu

- Fix* fixes the position of a vertex.
- Move* moves a vertex.
- Set Lengths* recalculates the edge lengths based on the settings chosen in the Set Edge Lengths Dialog Box.
- Randomize* randomizes the positions and/or velocities using the Randomize Vertices Dialog Box.
- Paste* puts a copy of the drawing onto the clipboard.

##### *Calculation* menu

- Initialize* prepares a new calculation.
- Start* starts the vertex position calculation. This opens the Calculation Control Dialog Box.
- Stop* stops the vertex position calculation.
- Automatic* lays out the graph without user intervention.

##### *Help* menu

- Contents* shows the help file contents.
- About* displays the version and copyright information.

---

## Set Edge Lengths Dialog Box

The *Set Edge Lengths* dialog box is used to specify how the edge lengths will be recalculated. The length of the edge is based on the count of edges connected to the edge's endpoints. Use the list box to choose how to combine these counts into a length.

Press the **OK** button to perform the recalculation, or **Cancel** to abort the operation.

---

## Randomize Vertices Dialog Box

The *Randomize Vertices* dialog box is used to specify the magnitude of the random perturbation to be applied to the **Position** and **Velocity** of each vertex that is not fixed.

Press the **OK** button to perform the randomization, or **Cancel** to abort the operation.

---

## Calculation Control Dialog Box

The *Calculation Control* dialog box is used to specify the **Time Step** for evolving the vertex positions, the **Temperature** and **Damping** parameters in the equation of motion, and the **A** and **B** parameters in the spring equation.

Check the **Pause** box to freeze the calculation. The display will be updated as long as the **Redraw** box is checked.

---

## File Formats

The *Vertex File* must be a tab-delimited text file ending with a blank line. Each line represents a vertex. The fields it contains are x-position, y-position, and a flag (0 or 1) indicating whether the vertex is fixed.

The *Edge File* must be a tab-delimited text file ending with a blank line. Each line represents an edge. The fields it contains are the index of the first vertex

(counting from zero) in the vertex file, the index of the second vertex (counting from zero), and the distance between the vertices.

Only limited file format checking is made by the *Open* command.

---

## Layout Algorithm

The *Layout Algorithm* solves a mechanical model where each edge is represented by a spring. (Optionally, all vertices may be connected to one another with springs of length equal to the shortest-path distance between the respective vertices.) The springs have a spring constant given by the equation  $k = \exp(-A d) / d^B$  (where  $d$  is the distance between the vertices) and have damped velocities. The vertices are also subject to thermal random forces (as in Brownian motion), specified by the temperature.

## IX. Appendix: Source Code

### A. Graph Layout Class

#### 1. GraphLayout.h

```

// Graph Layout Tool
// Copyright © 1996
// Regents of the University of California
// All rights reserved

// B. W. Bush
// Energy and Environmental Analysis Group
// TSA-4, Mail Stop F604
// Los Alamos National Laboratory
// Los Alamos, NM 87545
// 505-667-6485 / bwb@lanl.gov

// $RCSfile: GraphLayout.h $
// $Revision: 2.3 $
// $Date: 1996/09/18 13:36:57 $

#ifndef LANL_GRAPH_LAYOUT
#define LANL_GRAPH_LAYOUT

// Include standard C++ library header files.
#include <map>
#include <vector>

// A graph layout contains a specification for a graph and
// its two-dimensional layout. Functions are provided to
// alter/optimize the aesthetic layout of the graph.
// The Layout Algorithm solves a mechanical model where
// each edge is represented by a spring. (Optionally, all
// vertices may be connected to one another with springs of
// length equal to the shortest-path distance between the
// respective vertices.) The springs have a spring constant
// given by the equation k = exp(- A d) / d^B (where d is the
// distance between the vertices) and are damped. The vertices
// are also subject to thermal random forces (as in Brownian
// motion).
class TGraphLayout
{
public:

    // There are several ways to calculate the edge length
    // from the count of edges at either end of the edge:
    enum EEdgeLengthStyle {

```

```

        kUni,           // unity
        kMin,           // the minimum
        kSqrtMin,       // the square root of the minimum
        kMax,           // the maximum
        kSqrtMax,       // the square root of the maximum
        kSum,           // the sum
        kSqrtSum,       // the square root of the sum
        kSqrtSumSqr,    // the square root of the sum of the
        squares
        kHarmSum,       // the harmonic sum
        kSqrtHarmSum   // the square root of the harmonic
        sum
    };

    // Construct an instance.
    TGraphLayout();

    // Set the number of vertices. Return whether the
    // operation succeeded.
    bool SetVertexCount(short count = 0);

    // Set the position of the specified vertex to (x, y),
    // making it fixed in the layout if indicated. Return whether
    // the operation succeeded.
    bool SetVertexPosition(short vertex, double x, double y,
        bool fixed = false);

    // Connect the specified vertices with an edge of the
    // specified length. Return whether the operation succeeded.
    bool ConnectVertices(short fromVertex, short toVertex,
        double length = 1);

    // Set the lengths of the edges based on the specified
    // edge length style. Return whether the operation succeeded.
    bool SetEdgeLengths(EEEdgeLengthStyle style = kSqrtMin);

    // Get the number of vertices. Return whether the
    // operation succeeded.
    bool GetVertexCount(short& count) const;

    // Get the position of the specified vertex and whether
    // it is fixed in the layout. Return whether the operation
    // succeeded.
    bool GetVertexPosition(short vertex, double& x, double&
        y, bool& fixed) const;

    // Get the velocity of the specified vertex. Return
    // whether the operation succeeded.
    bool GetVertexVelocity(short vertex, double& vx, double&
        vy) const;

    // Get the kinetic and potential energy of the layout.
    // Return whether the operation succeeded.

```

```

    bool GetEnergy(double& kinetic, double& potential)
const;

    // Set the time step for the calculation to the
specified value. Return whether the operation succeeded.
    bool SetTimeStep(double timeStep = 0.1);

    // Set the temperature for the calculation to the
specified value. Return whether the operation succeeded.
    bool SetTemperature(double temperature = 0);

    // Set the damping factor for the calculation to the
specified value. Return whether the operation succeeded.
    bool SetDamping(double damping = 1);

    // Set the spring parameters for the calculation to the
specified value. Return whether the operation succeeded.
    bool SetSpringParameters(double a = 0, double b = 1);

    // Get the time step for the calculation.
    void GetTimeStep(double& timeStep) const;

    // Get the temperature for the calculation.
    void GetTemperature(double& temperature) const;

    // Get the damping factor for the calculation.
    void GetDamping(double& damping) const;

    // Get the spring parameters for the calculation.
    void GetSpringParameters(double& a, double& b) const;

    // Initialize the calculation, connecting all of the
vertices with springs if indicated. Return whether the
operation succeeded.
    bool Initialize(bool connectAll = true);

    // Return whether the calculation has been initialized.
    bool IsInitialized() const;

    // Randomize the positions of the vertices up to a
maximum of the bound. Return whether the operation
succeeded.
    bool RandomizePositions(double bound = 1);

    // Randomize the velocities of the vertices up to a
maximum of the bound. Return whether the operation
succeeded.
    bool RandomizeVelocities(double bound = 1);

    // Evolve the vertex positions the specified number of
time steps. Return whether the operation succeeded.
    bool EvolvePositions(long steps = 1);

        // Lay out the vertices automatically using the
specified absolute and relative tolerances for the kinetic
energy per vertex. Return whether the absolute tolerance
was met.
    bool LayoutAutomatically(double epsilonAbsolute = 1e-4,
double epsilonRelative = 1e-8);

private:

    // Use the standard namespace.
using namespace std;

    // Geometric point.
class TPoint
{
public:

    // Construct an instance with the specified
abscissa and ordinate.
    TPoint(double x = 0, double y = 0)
        : fX(x),
          fY(y)
    {
    }

    // Return the abscissa.
    double X() const
    {
        return fX;
    }

    // Return the ordinate.
    double Y() const
    {
        return fY;
    }

    // Shift the instance's location by adding the
specified point treated as a vector.
    TPoint& operator+=(const TPoint& point)
    {
        fX += point.fX;
        fY += point.fY;
        return *this;
    }

    // Shift the instance's location by subtracting the
specified point treated as a vector.
    TPoint& operator-=(const TPoint& point)
    {
        fX -= point.fX;
        fY -= point.fY;
        return *this;
    }
}

```

```

    // Scale the instance's location by multiplying it
by the specified scalar.
TPoint& operator*=(double number)
{
    fX *= number;
    fY *= number;
    return *this;
}

    // Scale the instance's location by dividing it by
the specified scalar.
TPoint& operator/=(double number)
{
    fX /= number;
    fY /= number;
    return *this;
}

    // Return the result of adding the specified point
treated as a vector to the instance.
TPoint operator+(const TPoint& point) const
{
    return TPoint(fX + point.fX, fY + point.fY);
}

    // Return the result of subtracting the specified
point treated as a vector from the instance.
TPoint operator-(const TPoint& point) const
{
    return TPoint(fX - point.fX, fY - point.fY);
}

    // Return the result of multiplying the instance by
the specified scalar.
TPoint operator*(double number) const
{
    return TPoint(fX * number, fY * number);
}

    // Return the result of dividing the instance by
the specified scalar.
TPoint operator/(double number) const
{
    return TPoint(fX / number, fY / number);
}

    // Return the dot product of the instance and the
specified point, both treated as vectors.
double operator%(const TPoint& point) const
{
    return fX * point.fX + fY * point.fY;
}

```

```

private:

    // Each instance has an abscissa.
    double fX;

    // Each instance has an ordinate.
    double fY;
};

    // Return whether the specified vertex count is valid.
bool ValidVertexCount(short vertexCount) const
{
    return vertexCount >= 0;
}

    // Return whether the specified vertex number is valid.
bool ValidVertex(short vertex) const
{
    return vertex >= 0 && vertex < fVertexCount;
}

    // Return the vector offset for the given matrix row
and column.
size_t Cell(short i, short j) const
{
    return i * fVertexCount + j;
}

    // Each instance has a vertex count.
short fVertexCount;

    // Each instance has a vector of vertex positions.
vector<TPoint> fPosition;

    // Each instance has a vector of vertex velocities.
vector<TPoint> fVelocity;

    // Each instance has a vector of flags indicating
whether the vertex position is fixed.
vector<bool> fFixed;

    // Each instance has a vector used to store the
adjacency matrix for the graph.
map<size_t, double, less<size_t> > fAdjacency;

    // Each instance has a flag indicating whether the
calculation has been initialized.
bool fInitialized;

    // Each instance has a vector used to store the
adjacency matrix for the springs.
map<size_t, double, less<size_t> > fDistance;

    // Each instance has a kinetic energy for its graph.

```

```

double fKinetic;

// Each instance has a potential energy for its graph.
double fPotential;

// Each instance has a time step for its calculation.
double fTimeStep;

// Each instance has a temperature for its calculation.
double fTemperature;

// Each instance has a damping factor for its
calculation.
double fDamping;

// Each instance has spring constants for its
calculation.
double fA, fB;

// A number representing infinity is defined.
static const double kInfinity;
};

#endif // LANL_GRAPH_LAYOUT

```

```

#include "GraphLayout.h"

// A number representing infinity is defined.
const double TGraphLayout::kInfinity = 1e20;

// Construct an instance.
TGraphLayout::TGraphLayout()
: fVertexCount(0),
fInitialized(false),
fTimeStep(0.1),
fTemperature(0),
fDamping(1),
fA(0),
fB(1)
{
}

// Set the number of vertices. Return whether the
operation succeeded.
bool TGraphLayout::SetVertexCount(short vertexCount)
{
    // Make sure the vertex count is valid.
    if (!ValidVertexCount(vertexCount))
        return false;

    // Set the vertex count and clear the graph.
    fVertexCount = vertexCount;
    fPosition = vector<TPoint>(fVertexCount);
    fVelocity = vector<TPoint>(fVertexCount);
    fFixed = vector<bool>(fVertexCount);
    fAdjacency = map<size_t, double, less<size_t> >();
    fInitialized = false;

    // Randomize the initial vertex positions.
    for (short i = 0; i < fVertexCount; ++i) {
        const double r = i % 10 + 1;
        const double theta = i;
        fPosition[i] = TPoint(r * cos(theta), r *
sin(theta));
        fFixed[i] = false;
        fAdjacency[Cell(i, i)] = 0;
    }

    // Return success.
    return true;
}

// Set the position of the specified vertex to (x, y),
making it fixed in the layout if indicated. Return whether
the operation succeeded.
bool TGraphLayout::SetVertexPosition(short vertex, double x,
double y, bool fixed)
{
    // Make sure the vertex number is valid.
}

```

## 2. GraphLayout.cpp

```

// Graph Layout Tool
// Copyright © 1996
// Regents of the University of California
// All rights reserved

// B. W. Bush
// Energy and Environmental Analysis Group
// TSA-4, Mail Stop F604
// Los Alamos National Laboratory
// Los Alamos, NM 87545
// 505-667-6485 / bwb@lanl.gov

// $RCSfile: GraphLayout.cpp $
// $Revision: 2.3 $
// $Date: 1996/09/18 14:16:18 $

// Include standard C++ library header files.
#include <map>
#include <math>
#include <stdlib>
#include <vector>

// Include graph layout tool header files.

```

```

if (!ValidVertex(vertex))
    return false;

// Set the vertex position.
fPosition[vertex] = TPoint(x, y);

// Clear the vertex velocity if the vertex is fixed.
if (fixed)
    fVelocity[vertex] = TPoint(0, 0);
fFixed[vertex] = fixed;

// Return success.
return true;
}

// Connect the specified vertices with an edge of the
// specified length. Return whether the operation succeeded.
bool TGraphLayout::ConnectVertices(short fromVertex, short
toVertex, double length)
{
    // Make sure the vertex numbers are valid.
    if (!ValidVertex(fromVertex) || !ValidVertex(toVertex))
        return false;

    // Connect the vertices.
    fAdjacency[Cell(fromVertex, toVertex)] = length;
    fAdjacency[Cell(toVertex, fromVertex)] = length;

    // The calculation will need to be initialized since a
length has changed.
    fInitialized = false;

    // Return success.
    return true;
}

// Set the lengths of the edges based on the specified edge
length style. Return whether the operation succeeded.
bool TGraphLayout::SetEdgeLengths(TGraphLayout::EEdgeLengthStyle
style)
{
    // Find the number of edges at each vertex.
    vector<short> connectionCount(fVertexCount);
    for (short i = 0; i < fVertexCount; ++i) {
        connectionCount[i] = 0;
        for (short j = 0; j < fVertexCount; ++j)
            if (fAdjacency.find(Cell(i, j)) !=
fAdjacency.end() && fAdjacency[Cell(i, j)] > 0)
                connectionCount[i] += (short) 1;
    }

    // Set the edge lengths based on the number of edges at
the end of each edge.

    for (short i = 0; i < fVertexCount; ++i)
        for (short j = 0; j < fVertexCount; ++j)
            if (fAdjacency.find(Cell(i, j)) !=
fAdjacency.end() && fAdjacency[Cell(i, j)] > 0)
                switch (style) {
                    case kUni:
                        fAdjacency[Cell(i, j)] = 1;
                        break;
                    case kMin:
                        fAdjacency[Cell(i, j)] =
::min(connectionCount[i], connectionCount[j]);
                        break;
                    case kSqrtMin:
                        fAdjacency[Cell(i, j)] =
sqrt(::min(connectionCount[i], connectionCount[j]));
                        break;
                    case kMax:
                        fAdjacency[Cell(i, j)] =
::max(connectionCount[i], connectionCount[j]);
                        break;
                    case kSqrtMax:
                        fAdjacency[Cell(i, j)] =
sqrt(::max(connectionCount[i], connectionCount[j]));
                        break;
                    case kSum:
                        fAdjacency[Cell(i, j)] =
connectionCount[i] + connectionCount[j];
                        break;
                    case kSqrtSum:
                        fAdjacency[Cell(i, j)] =
sqrt(connectionCount[i] + sqrt(connectionCount[j]));
                        break;
                    case kSqrtSumSqr:
                        fAdjacency[Cell(i, j)] =
sqrt(pow(connectionCount[i], 2) + pow(connectionCount[j],
2));
                        break;
                    case kHarmSum:
                        fAdjacency[Cell(i, j)] =
1 / (1 / (double) connectionCount[i] + 1 / (double)
connectionCount[j]);
                        break;
                    case kSqrtHarmSum:
                        fAdjacency[Cell(i, j)] =
1 / sqrt(1 / (double) connectionCount[i] + 1 / (double)
connectionCount[j]);
                        break;
                }
}

// The calculation will need to be initialized since a
length has changed.
fInitialized = false;

// Return success.

```

```

        return true;
    }

    // Get the number of vertices.  Return whether the
    // operation succeeded.
    bool TGraphLayout::GetVertexCount(short& count) const
    {
        // Copy the number of vertices.
        count = fVertexCount;

        // Return whether the number of vertices is valid.
        return ValidVertexCount(fVertexCount);
    }

    // Get the position of the specified vertex and whether it
    // is fixed in the layout.  Return whether the operation
    // succeeded.
    bool TGraphLayout::GetVertexPosition(short vertex, double&
x, double& y, bool& fixed) const
    {
        // Make sure the vertex number is valid.
        if (!ValidVertex(vertex))
            return false;

        // Get the vertex position and is fixed state.
        x = fPosition[vertex].X();
        y = fPosition[vertex].Y();
        fixed = fFixed[vertex];

        // Return success.
        return true;
    }

    // Get the velocity of the specified vertex.  Return
    // whether the operation succeeded.
    bool TGraphLayout::GetVertexVelocity(short vertex, double&
vx, double& vy) const
    {
        // Make sure the vertex number is valid.
        if (!ValidVertex(vertex))
            return false;

        // Get the vertex velocity.
        vx = fVelocity[vertex].X();
        vy = fVelocity[vertex].Y();

        // Return success.
        return true;
    }

    // Get the kinetic and potential energy of the layout.
    // Return whether the operation succeeded.
    bool TGraphLayout::GetEnergy(double& kinetic, double&
potential) const
    {
        // Make sure the calculation is initialized.
        if (!IsInitialized())
            return false;

        // Get the energies.
        kinetic = fKinetic;
        potential = fPotential;

        // Return sucess.
        return true;
    }

    // Set the time step for the calculation to the specified
    // value.  Return whether the operation succeeded.
    bool TGraphLayout::SetTimeStep(double timeStep)
    {
        // Make sure the time step is valid.
        if (!(timeStep > 0))
            return false;

        // Set the time step.
        fTimeStep = timeStep;

        // Return success.
        return true;
    }

    // Set the temperature for the calculation to the specified
    // value.  Return whether the operation succeeded.
    bool TGraphLayout::SetTemperature(double temperature)
    {
        // Make sure the temperature is valid.
        if (!(temperature >= 0))
            return false;

        // Set the temperature.
        fTemperature = temperature;

        // Return success.
        return true;
    }

    // Set the damping factor for the calculation to the
    // specified value.  Return whether the operation succeeded.
    bool TGraphLayout::SetDamping(double damping)
    {
        // Make sure the damping factor is valid.
        if (!(damping > 0))
            return false;

        // Set the damping factor.
        fDamping = damping;
    }
}

```

```

    // Return success.
    return true;
}

// Set the spring parameters for the calculation to the
// specified value. Return whether the operation succeeded.
bool TGraphLayout::SetSpringParameters(double a, double b)
{
    // Set the spring parameters.
    fA = a;
    fB = b;

    // Return success.
    return true;
}

// Get the time step for the calculation.
void TGraphLayout::GetTimeStep(double& timeStep) const
{
    // Get the time step.
    timeStep = fTimeStep;
}

// Get the temperature for the calculation.
void TGraphLayout::GetTemperature(double& temperature) const
{
    // Get the temperature.
    temperature = fTemperature;
}

// Get the damping factor for the calculation.
void TGraphLayout::GetDamping(double& damping) const
{
    // Get the damping factor.
    damping = fDamping;
}

// Get the spring parameters for the calculation.
void TGraphLayout::GetSpringParameters(double& a, double& b) const
{
    // Get the spring parameters.
    a = fA;
    b = fB;
}

// Initialize the calculation, connecting all of the
// vertices with springs if indicated. Return whether the
// operation succeeded.
bool TGraphLayout::Initialize(bool connectAll)
{
    // Start with the distances the same as the
    // adjacencies.
    fDistance = fAdjacency;
}

```

```

    // Connect each vertex to every other vertex if
    // necessary.
    if (connectAll) {

        // Fill the matrix.
        for (short i = 0; i < fVertexCount; ++i)
            for (short j = 0; j < i; ++j)
                if (fDistance.find(Cell(i, j)) ==
fDistance.end())
                    fDistance[Cell(i, j)] = kInfinity;
                    fDistance[Cell(j, i)] = kInfinity;
                }

        // Construct the shortest path spanning tree.
        double maximumDistance = 0;
        for (bool changed = true; changed; ) {
            map<size_t, double, less<size_t> > oldDistance =
fDistance;
            changed = false;
            for (short i = 0; i < fVertexCount; ++i)
                for (short j = 0; j < i; ++j) {
                    const double previousDistance =
oldDistance[Cell(i, j)];
                    double distance = previousDistance;
                    for (short k = 0; k < fVertexCount; ++k)
                        distance = ::min(distance,
oldDistance[Cell(i, k)] + oldDistance[Cell(k, j)]);
                    if (distance == kInfinity)
                        continue;
                    maximumDistance = ::max(maximumDistance,
distance);
                    fDistance[Cell(i, j)] = distance;
                    fDistance[Cell(j, i)] = distance;
                    changed |= distance != previousDistance;
                }
            }

        // Convert infinite distances to finite ones.
        maximumDistance *= 2;
        for (short i = 0; i < fVertexCount; ++i)
            for (short j = 0; j < fVertexCount; ++j)
                if (fDistance[Cell(i, j)] == kInfinity)
                    fDistance[Cell(i, j)] = maximumDistance;
            }

        // The calculation is initialized now.
        fInitialized = true;
        fKinetic = 0;
        fPotential = 0;

        // Return success.
        return true;
    }

```

```

}

// Return whether the calculation has been initialized.
bool TGraphLayout::IsInitialized() const
{
    // Return the initialization state.
    return fInitialized;
}

// Randomize the positions of the vertices up to a maximum
of the bound. Return whether the operation succeeded.
bool TGraphLayout::RandomizePositions(double bound)
{
    // Make sure the bound is valid.
    if (!(bound >= 0))
        return false;

    // Randomly move the vertex positions for the vertices
    // that are not fixed
    for (short i = 0; i < fVertexCount; ++i) {
        if (fFixed[i])
            continue;
        const double r = bound * rand() / RAND_MAX;
        const double theta = 2 * M_PI * rand() / RAND_MAX;
        fPosition[i] += TPoint(r * cos(theta), r *
sin(theta));
    }

    // Return success.
    return true;
}

// Randomize the velocities of the vertices up to a maximum
of the bound. Return whether the operation succeeded.
bool TGraphLayout::RandomizeVelocities(double bound)
{
    // Make sure the bound is valid.
    if (!(bound >= 0))
        return false;

    // Randomly alter the vertex velocities for the
    // vertices that are not fixed.
    for (short i = 0; i < fVertexCount; ++i) {
        if (fFixed[i])
            continue;
        const double r = bound * rand() / RAND_MAX;
        const double theta = 2 * M_PI * rand() / RAND_MAX;
        fVelocity[i] += TPoint(r * cos(theta), r *
sin(theta));
    }

    // Return success.
    return true;
}

// Evolve the vertex positions the specified number of time
steps. Return whether the operation succeeded.
bool TGraphLayout::EvolvePositions(long steps)
{
    // Make sure the calculation has been initialized and
    that the number of steps is valid.
    if (!IsInitialized() || !(steps > 0))
        return false;

    // Repeat for each step.
    for (long step = 0; step < steps; ++step) {

        // Keep track of the accelerations and energies.
        vector<TPoint> acceleration(fVertexCount);
        vector<double> kssq(fVertexCount);
        fKinetic = 0;
        fPotential = 0;

        // Repeat initialization for each vertex.
        for (short i = 0; i < fVertexCount; ++i) {

            // Ignore fixed vertices.
            if (fFixed[i])
                continue;

            // Clear the acceleration and damping
            multipliers.
            acceleration[i] = 0;
            kssq[i] = 0;
        }

        // Calculate matrix forces.
        for (map<size_t, double, less<size_t> >::iterator
iter = fDistance.begin(); iter != fDistance.end(); ++iter) {
            const short i = (short) ((*iter).first /
fVertexCount);
            const short j = (short) ((*iter).first %
fVertexCount);
            const double distance = (*iter).second;
            if (distance == 0 || distance == kInfinity)
                continue;
            const TPoint delta = fPosition[j] -
fPosition[i];
            const double deltaNorm = sqrt(delta % delta + 1
/ kInfinity);
            const double displacement = deltaNorm -
distance;
            const double k = exp(- fA * distance) /
pow(distance, fB);
            kssq[i] += k * k;
            acceleration[i] += delta * (k * displacement /
deltaNorm);
        }
    }
}

```

```

2;                                fPotential += k * displacement * displacement /
}                                }

// Calculate vector forces.
for (short i = 0; i < fVertexCount; ++i) {
    // Ignore fixed vertices.
    if (fFixed[i])
        continue;

    // Damp velocities.
    const double damping = sqrt(kssq[i]) * fDamping;
    acceleration[i] -= fVelocity[i] * damping;

    // Add the acceleration due to the random
    force, if any.
    if (fTemperature > 0) {
        const double r = sqrt(- 4 * fTemperature *
damping * log((rand() + 1.) / (RAND_MAX + 1.)));
        const double theta = 2 * M_PI * rand() / 
RAND_MAX;
        acceleration[i] += TPoint(r * cos(theta), r
* sin(theta)) * fTimeStep;
    }

    // Update the positions.
    fPosition[i] += (fVelocity[i] + acceleration[i]
* (fTimeStep / 2)) * fTimeStep;
    fVelocity[i] += acceleration[i] * fTimeStep;
    fKinetic += (fVelocity[i] % fVelocity[i]) / 2;
}

// Return success.
return true;
}

// Lay out the vertices automatically using the specified
absolute and relative tolerances for the kinetic energy per
vertex. Return whether the absolute tolerance was met.
bool          TGraphLayout::LayoutAutomatically(double
epsilonAbsolute, double epsilonRelative)
{
    // Set the initial parameters.
    double kinetic, potential, delta;
    SetTimeStep(0.1);
    SetDamping(1.0);
    SetTemperature(1.0);
    SetSpringParameters(0.0, 1.0);
    EvolvePositions(10);
    GetEnergy(kinetic, potential);

    // Perform the first stage of the evolution to organize
    the large-scale structure of the graph.
    SetTemperature(0.0);
    SetSpringParameters(0.0, 0.0);
    do {
        EvolvePositions(1);
        GetEnergy(kinetic, potential);
    } while (kinetic / fVertexCount > epsilonAbsolute);

    // Perform the second stage of the evolution to organize
    the small-scale structure of the graph.
    SetSpringParameters(1.0, -1.0);
    EvolvePositions(10);
    GetEnergy(kinetic, potential);
    do {
        delta = kinetic;
        EvolvePositions(1);
        GetEnergy(kinetic, potential);
        delta -= kinetic;
    } while (kinetic / fVertexCount > epsilonAbsolute &&
fabs(delta / fVertexCount) > epsilonRelative);

    // Return whether the absolute tolerance was met.
    return kinetic / fVertexCount <= epsilonAbsolute;
}

```

## B. Command Line Tool

### 1. Main.cpp

```

// Graph Layout Tool
// Copyright © 1996
// Regents of the University of California
// All rights reserved

// B. W. Bush
// Energy and Environmental Analysis Group
// TSA-4, Mail Stop F604
// Los Alamos National Laboratory
// Los Alamos, NM 87545
// 505-667-6485 / bwb@lanl.gov

// $RCSfile: Main.cpp $
// $Revision: 2.1 $
// $Date: 1996/03/18 09:03:51 $

// Include standard C++ library header files.
#include <iostream>
#include <iomanip>
#include <iostream>

```

```

#include <string>
#include <iostream>
#include <vector>

// Use the standard namespace.
using namespace std;

// Include graph layout tool header files.
#include "GraphLayout.h"

// Entry point.
int main(int argc, char* argv[])
{
    // Set defaults.
    bool error = false;
    bool help = false;
    string vertexFilename = "";
    string edgeFilename = "";
    string outputFilename = "";
    long iterations = 1000;
    double timeStep = 0.01;
    double temperature = 0;
    double damping = 1;
    double a = 0;
    double b = 1;
    bool connectAll = false;
    bool ignoreLengths = false;
    bool automatic = false;

    // Interpret command line options.
    for (int i = 1; i < argc; ++i) {
        const std::string arg(argv[i]);
        if (arg == "-h")
            help = true;
        else if (arg == "-v")
            vertexFilename = argv[++i];
        else if (arg == "-e")
            edgeFilename = argv[++i];
        else if (arg == "-o")
            outputFilename = argv[++i];
        else if (arg == "-i") {
            istrstream data(argv[+i]);
            data >> iterations;
        } else if (arg == "-s") {
            istrstream data(argv[+i]);
            data >> timeStep;
        } else if (arg == "-t") {
            istrstream data(argv[+i]);
            data >> temperature;
        } else if (arg == "-d") {
            istrstream data(argv[+i]);
            data >> damping;
        } else if (arg == "-k") {
            istrstream data1(argv[+i]);
            data1 >> a;
            istrstream data2(argv[+i]);
            data2 >> b;
        } else if (arg == "-a")
            connectAll = true;
        else if (arg == "-x")
            ignoreLengths = true;
        else if (arg == "-A")
            automatic = true;
        else {
            cerr << "Invalid argument: " << arg << endl;
            error = true;
        }
    }

    // Display help.
    if (help) {
        cerr << "Usage: " << endl;
        cerr << " GraphLayoutCL [-h] -v filename -e
filename -o filename [-i number]" << endl;
        cerr << " [-s number] [-t number] [-d
number] [-k number number]" << endl;
        cerr << " [-a] [-A]" << endl;
        cerr << "Parameters:" << endl;
        cerr << " -h display help" << endl;
        cerr << " -v vertex filename" << endl;
        cerr << " -e edge filename" << endl;
        cerr << " -o output filename" << endl;
        cerr << " -i number of iterations [default =
1000]" << endl;
        cerr << " -s time step [default = 0.01]" << endl;
        cerr << " -t temperature [default = 0]" << endl;
        cerr << " -d damping [default = 1]" << endl;
        cerr << " -k spring parameters [default = 0 1]" <<
endl;
        cerr << " -a connect all vertices with springs
[default = no]" << endl;
        cerr << " -x ignore lengths in edge file and use
optimized edge lengths instead" << endl;
        cerr << " -A proceed automatically until
convergence (parameters istdk ignored)" << endl;
        cerr << "File formats:" << endl;
        cerr << " The Vertex File must be a tab-delimited
text file ending with a blank line." << endl;
        cerr << " Each line represents a vertex. The
fields it contains are x-position," << endl;
        cerr << " y-position, and a flag (0 or 1)
indicating whether the vertex is fixed." << endl;
        cerr << " The Edge File must be a tab-delimited
text file ending with a blank line." << endl;
        cerr << " Each line represents an edge. The fields
it contains are the index of the" << endl;
        cerr << " first vertex (counting from zero) in the
vertex file, the index of the second" << endl;
    }
}

```

```

        cerr << "    vertex (counting from zero), and the
distance between the vertices." << endl;
        cerr << "Algorithm:" << endl;
        cerr << "    The Layout Algorithm solves a mechanical
model where each edge is represented" << endl;
        cerr << "    by a spring. (Optionally, all vertices
may be connected to one another with)" << endl;
        cerr << "    springs of length equal to the shortest-
path distance between the respective" << endl;
        cerr << "    vertices.) The springs have a spring
constant given by the equation k = " << endl;
        cerr << "    exp(- A d) / d^B (where d is the distance
between the vertices) and are" << endl;
        cerr << "    damped. The vertices are also subject to
thermal random forces (as in" << endl;
        cerr << "    Brownian motion)." << endl;
        cerr << "Contact:" << endl;
        cerr << "    B. W. Bush, Energy and Environmental
Analysis Group, TSA-4," << endl;
        cerr << "    Mail Stop F604, Los Alamos National
Laboratory, Los Alamos, NM 87545" << endl;
        cerr << "    505-667-6485 / bwb@lanl.gov" << endl;
        cerr << "Copyright 1996 Regents of the University of
California, All rights reserved." << endl;
        cerr << endl;
        return 0;
    }

    // Make sure file names were specified.
    if (vertexFilename == "") {
        cerr << "Vertex filename not specified." << endl;
        error = true;
    }
    if (edgeFilename == "") {
        cerr << "Edge filename not specified." << endl;
        error = true;
    }
    if (outputFilename == "") {
        cerr << "Output filename not specified." << endl;
        error = true;
    }

    // Abort if the command line contained an error.
    if (error) {
        cerr << endl;
        return -1;
    }

    // Display settings.
    cerr << "Settings:" << endl;
    cerr << "    Vertex filename: " << vertexFilename << endl;
    cerr << "    Edge filename: " << edgeFilename << endl;
    cerr << "    Output filename: " << outputFilename << endl;
    if (!automatic) {

```

```

        cerr << "    Number of iterations: " << iterations <<
endl;
        cerr << "    Time step: " << timeStep << endl;
        cerr << "    Temperature: " << temperature << endl;
        cerr << "    Damping: " << damping << endl;
        cerr << "    Spring parameters: A = " << a << ", B = "
<< b << endl;
    }
    cerr << "    Connect all vertices: " << (connectAll ?
"yes" : "no") << endl;
    cerr << "    Ignore edge lengths: " << (ignoreLengths ?
"yes" : "no") << endl;
    cerr << "    Automatic: " << (automatic ? "yes" : "no") <<
endl;

    // Initialize graph.
    short vertexCount = -1;
    short edgeCount = -1;
    TGraphLayout layout;

    // Read input files.
    {
        char line[1000];
        for (ifstream vertexFile(vertexFilename.c_str());
!vertexFile.eof() && !vertexFile.bad();
vertexFile.getline(line, sizeof(line)))
            ++vertexCount;
        for (ifstream edgeFile(edgeFilename.c_str());
!edgeFile.eof() && !edgeFile.bad(); edgeFile.getline(line,
sizeof(line)))
            ++edgeCount;
        layout.SetVertexCount(vertexCount);
        cerr << "    Vertices: " << vertexCount << endl;
        cerr << "    Edges: " << edgeCount << endl;
        cerr << endl;
        ifstream vertexFile(vertexFilename.c_str());
        for (short i = 0; i < vertexCount; ++i) {
            double x, y;
            bool fixed;
            vertexFile >> x >> y >> fixed;
            layout.SetVertexPosition(i, x, y, fixed);
        }
        if (vertexFile.bad()) {
            cerr << "Could not read vertices." << endl;
            cerr << endl;
            return -1;
        }
        ifstream edgeFile(edgeFilename.c_str());
        for (short i = 0; i < edgeCount; ++i) {
            short from, to;
            double length;
            edgeFile >> from >> to >> length;
            layout.ConnectVertices(from, to, length);
        }
    }

```

```

        if (edgeFile.bad()) {
            cerr << "Could not read edges." << endl;
            cerr << endl;
            return -1;
        }
    }

    // Set lengths if necessary.
    if (ignoreLengths)
        layout.SetEdgeLengths();
    layout.Initialize(connectAll);

    // Perform calculation.
    if (!automatic) {
        layout.SetTimeStep(timeStep);
        layout.SetTemperature(temperature);
        layout.SetDamping(damping);
        layout.SetSpringParameters(a, b);
        layout.EvolvePositions(iterations);
    } else {
        if (!layout.LayoutAutomatically())
            cerr << "Steady state reached before\r\nkinetic
energy damped out." << endl;
    }

    // Write output file.
    {
        ofstream vertexFile(outputFilename.c_str());
        for (short i = 0; i < vertexCount; ++i) {
            double x, y;
            bool fixed;
            layout.GetVertexPosition(i, x, y, fixed);
            vertexFile << setprecision(20) << x << "\t" <<
setprecision(20) << y << "\t" << fixed << endl;
        }
    }

    // Exit.
    return 0;
}

```

## C. Graphical Tool

### 1. OwlMain.cpp

```

// Graph Layout Tool
// Copyright © 1996
// Regents of the University of California
// All rights reserved

// B. W. Bush

```

```

// Energy and Environmental Analysis Group
// TSA-4, Mail Stop F604
// Los Alamos National Laboratory
// Los Alamos, NM 87545
// 505-667-6485 / bwb@lanl.gov

// $RCSfile: OwlMain.cpp $
// $Revision: 2.0 $
// $Date: 1996/03/06 10:13:35 $

// Include standard C++ library header files.
#include <string>

// Include OWL header files.
#include <owl\pch>

// Include graph layout tool header files.
#include "GraphLayoutApplication.h"

// Entry point.
int OwlMain(int, char*[])
{
    return TGraphLayoutApplication().Run();
}

```

### 2. GraphLayoutApplication.h

```

// Graph Layout Tool
// Copyright © 1996
// Regents of the University of California
// All rights reserved

// B. W. Bush
// Energy and Environmental Analysis Group
// TSA-4, Mail Stop F604
// Los Alamos National Laboratory
// Los Alamos, NM 87545
// 505-667-6485 / bwb@lanl.gov

// $RCSfile: GraphLayoutApplication.h $
// $Revision: 2.0 $
// $Date: 1996/03/06 10:13:35 $


```

```

#ifndef LANL_GRAPH_LAYOUT_APPLICATION
#define LANL_GRAPH_LAYOUT_APPLICATION

// Include standard C++ library header files.
#include <string>
```

```

// Include OWL header files.
#include <owl\pch>
#include <owl\application>

// A graph layout application class.
class TGraphLayoutApplication
: public TApplication
{
public:
    // Initialize the main window.
    void InitMainWindow();
};

#endif // LANL_GRAPH_LAYOUT_APPLICATION

// Graph Layout Tool
// Copyright © 1996
// Regents of the University of California
// All rights reserved

// B. W. Bush
// Energy and Environmental Analysis Group
// TSA-4, Mail Stop F604
// Los Alamos National Laboratory
// Los Alamos, NM 87545
// 505-667-6485 / bwb@lanl.gov

// RCSfile: GraphLayoutApplication.cpp $
// Revision: 2.3 $
// Date: 1996/09/18 13:33:11 $

// Include standard C++ library header files.
#include <string>

// Include OWL header files.
#include <owl\pch>
#include <owl\application>
#include <owl\butonga>
#include <owl\controlrb>
#include <owl\decframe>
#include <owl\docking>
#include <owl\statusba>

// Include graph layout tool header files.
#include "GraphLayoutWindow.h"
#include "GraphLayoutApplication.h"
#include "GraphLayout.rh"

// Initialize the main window.
void TGraphLayoutApplication::InitMainWindow()
{
    // Create the frame.
    TDecoratedFrame* main = new TDecoratedFrame(0, "Graph
Layout", new TGraphLayoutWindow, true);

    // Add a status bar to the frame.
    TStatusBar* sb = new TStatusBar(main,
TGadget::Recessed);
    main->Insert(*sb, TDecoratedFrame::Bottom);

    // Add a control bar to the frame.
    TDockableControlBar* cb = new TDockableControlBar(main);
    cb->SetHintMode(TGadgetWindow::EnterHints);
    cb->Insert(*new TButtonGadget(IDB_FILE_OPEN,
IDC_FILE_OPEN, TButtonGadget::Command, TRUE));
    cb->Insert(*new TButtonGadget(IDB_FILE_SAVE,
IDC_FILE_SAVE, TButtonGadget::Command, TRUE));
    cb->Insert(*new TSeparatorGadget());
    cb->Insert(*new TButtonGadget(IDB_EDIT_FIX,
IDC_EDIT_FIX, TButtonGadget::Command, TRUE));
    cb->Insert(*new TButtonGadget(IDB_EDIT_MOVE,
IDC_EDIT_MOVE, TButtonGadget::Command, TRUE));
    cb->Insert(*new TButtonGadget(IDB_EDIT_LENGTH,
IDC_EDIT_LENGTH, TButtonGadget::Command, TRUE));
    cb->Insert(*new TButtonGadget(IDB_EDIT_RANDOMIZE,
IDC_EDIT_RANDOMIZE, TButtonGadget::Command, TRUE));
    cb->Insert(*new TSeparatorGadget());
    cb->Insert(*new TButtonGadget(IDB_CALC_INITIALIZE,
IDC_CALC_INITIALIZE, TButtonGadget::Command, TRUE));
    cb->Insert(*new TButtonGadget(IDB_CALC_START,
IDC_CALC_START, TButtonGadget::Command, TRUE));
    cb->Insert(*new TButtonGadget(IDB_CALC_STOP,
IDC_CALC_STOP, TButtonGadget::Command, TRUE));
    cb->Insert(*new TButtonGadget(IDB_CALC_AUTO,
IDC_CALC_AUTO, TButtonGadget::Command, TRUE));
    cb->Insert(*new TSeparatorGadget());
    cb->Insert(*new TButtonGadget(IDB_HELP_CONTENTS,
IDC_HELP_CONTENTS, TButtonGadget::Command, TRUE));
    cb->Insert(*new TButtonGadget(IDB_HELP_ABOUT,
IDC_HELP_ABOUT, TButtonGadget::Command, TRUE));
    (new THarbor(*main))->Insert(*cb, alTop);

    // Add command accelerators to the frame.
    main->Attr.AccelTable = IDA_GRAPH_LAYOUT;

    // Add a menu to the frame.
    main->AssignMenu(IDM_GRAPH_LAYOUT);

    // Set the frame's icon.
    main->SetIcon(this, IDI_GRAPH_LAYOUT);
}

```

```

    // Make the frame the application's main window.
    SetMainWindow(main);
}

// Make the frame the application's main window.
SetMainWindow(main);

// Paint the window using the specified device context.
void Paint(TDC& dc, bool, TRect&);

// Respond to resizing.
void EvSize(uint, TSize&);

// Respond to a mouse button going down at the
specified location.
void EvLButtonDown(uint, TPoint& location);

// Respond to a mouse button going up at the specified
location.
void EvLButtonUp(uint, TPoint& location);

// Respond to a request to close the window.
bool CanClose();

// Process the open command.
void Open();

// Process the close command.
void Close();

// Process the save command.
void Save();

// Process the save as command.
void SaveAs();

// Process the fix position command.
void Fix();

// Process the move vertex command.
void Move();

// Process the set lengths command.
void SetLengths();

// Process the randomize command.
void Randomize();

// Process the paste command.
void Paste();

// Process the initialize command.
void Initialize();

// Process the start command.
void Start();

// Process the stop command.
void Stop();

// Process the automatic command.

```

#### 4. GraphLayoutWindow.h

```

// Graph Layout Tool
// Copyright © 1996
// Regents of the University of California
// All rights reserved

// B. W. Bush
// Energy and Environmental Analysis Group
// TSA-4, Mail Stop F604
// Los Alamos National Laboratory
// Los Alamos, NM 87545
// 505-667-6485 / bwb@lanl.gov

// $RCSfile: GraphLayoutWindow.h $
// $Revision: 2.1 $
// $Date: 1996/03/13 08:37:44 $

#ifndef LANL_GRAPH_LAYOUTWINDOW
#define LANL_GRAPH_LAYOUTWINDOW

// Include standard C++ library header files.
#include <string>
#include <vector>

// Include OWL header files.
#include <owl\pch>
#include <owl>window>

// Include graph layout tool header files.
#include "GraphLayout.h"
#include "Mapper.h"
#include "ControlPanelDialog.h"

// Graph layout main window.
class TGraphLayoutWindow
    : public TWindow
{
public:

    // Construct an instance with the specified parent.
    TGraphLayoutWindow(TWindow* parent = NULL);

    // Set up the window.
    void SetupWindow();

```

```

void Automatic();

// Process the help contents command.
void Help();

// Process the help about command.
void About();

// Process an enabling request for opening.
void CanOpen(TCommandEnabler&);

// Process an enabling request for editing.
void CanEdit(TCommandEnabler&);

// Process an enabling request for starting.
void CanStart(TCommandEnabler&);

// Process an enabling request for stoping.
void CanStop(TCommandEnabler&);

private:
    // Use the standard namespace.
    using namespace std;

    // The calculation may be in one of several states.
    enum EState {kClosed, kEditing, kFixing, kMoving,
    kRunning, kStopping};

    // Graph edge.
    class TEdge
    {
    public:
        // Construct an instance from and to the specified
        vertices.
        TEdge(short from = 0, short to = 0)
            : fFrom(from),
              fTo(to)
        {}

        // Return the from vertex.
        short From() const {return fFrom;}

        // Return the to vertex.
        short To() const {return fTo;}
    private:
        // Each instance has a from vertex.
        short fFrom;

        // Each instance has a to vertex.
    };

    short fTo;
};

// Find the vertex near a point, if any.
void FindVertex(const TPoint& location);

// Update the control panel.
void UpdateControlPanel();

// Compute the graph layout.
static DWORD WINAPI Compute(LPVOID);

// Each instance has a control panel dialog.
TControlPanelDialog fControl;

// Each instance has a screen mapper.
TScreenMapper<double, double> fMap;

// Each instance has a vertex file name.
::string fFilename;

// Each instance is in a calculation state.
EState fState;

// Each instance has a count of vertices.
short fVertexCount;

// Each instance has a count of edges.
short fEdgeCount;

// Each instance has a vector of edges.
vector<TEdge> fEdges;

// Each instance has a layout.
TGraphLayout fLayout;

// Each instance has a calculational time.
double fTime;

// Each instance has a calculation increment size.
int fIncrement;

// Each instance has a vertex selection.
short fVertex;

// Declare response table.
DECLARE_RESPONSE_TABLE(TGraphLayoutWindow);
};

#endif // LANL_GRAPH_LAYOUT_WINDOW

```

## 5. GraphLayoutWindow.cpp

```

// Graph Layout Tool
// Copyright © 1996
// Regents of the University of California
// All rights reserved

// B. W. Bush
// Energy and Environmental Analysis Group
// TSA-4, Mail Stop F604
// Los Alamos National Laboratory
// Los Alamos, NM 87545
// 505-667-6485 / bwb@lanl.gov

// RCSfile: GraphLayoutWindow.cpp $
// $Revision: 2.3 $
// $Date: 1996/09/18 13:32:42 $

// Include standard C++ library header files.
#include <fstream>
#include <iomanip>
#include <stdio>
#include <string>

// Include OWL header files.
#include <owl\pch>
#include <owl\clipboar>
#include <owl\dc>
#include <owl\gdiobjec>
#include <owl\metafile>
#include <owl\opensave>

// Include graph layout tool header files.
#include "EdgeLengthDialog.h"
#include "RandomizerDialog.h"
#include "ControlPanelDialog.h"
#include "GraphLayoutWindow.h"
#include "GraphLayout.rh"

// Define the response table.
DEFINE_RESPONSE_TABLE1(TGraphLayoutWindow, TWindow)
    EV_WM_SIZE,
    EV_WM_LBUTTONDOWN,
    EV_WM_LBUTTONUP,
    EV_COMMAND(IDC_FILE_OPEN, Open),
    EV_COMMAND_ENABLE(IDC_FILE_OPEN, CanOpen),
    EV_COMMAND(IDC_FILE_CLOSE, Close),
    EV_COMMAND_ENABLE(IDC_FILE_CLOSE, CanEdit),
    EV_COMMAND(IDC_FILE_SAVE, Save),
    EV_COMMAND_ENABLE(IDC_FILE_SAVE, CanEdit),
    EV_COMMAND(IDC_FILE_SAVEAS, SaveAs),
    EV_COMMAND_ENABLE(IDC_FILE_SAVEAS, CanEdit),
    EV_COMMAND(IDC_EDIT_FIX, Fix),

```

```

    EV_COMMAND_ENABLE(IDC_EDIT_FIX, CanEdit),
    EV_COMMAND(IDC_EDIT_MOVE, Move),
    EV_COMMAND_ENABLE(IDC_EDIT_MOVE, CanEdit),
    EV_COMMAND(IDC_EDIT_LENGTH, SetLengths),
    EV_COMMAND_ENABLE(IDC_EDIT_LENGTH, CanEdit),
    EV_COMMAND(IDC_EDIT_RANDOMIZE, Randomize),
    EV_COMMAND_ENABLE(IDC_EDIT_RANDOMIZE, CanEdit),
    EV_COMMAND(IDC_EDIT_PASTE, Paste),
    EV_COMMAND_ENABLE(IDC_EDIT_PASTE, CanEdit),
    EV_COMMAND(IDC_CALC_INITIALIZE, Initialize),
    EV_COMMAND_ENABLE(IDC_CALC_INITIALIZE, CanEdit),
    EV_COMMAND(IDC_CALC_START, Start),
    EV_COMMAND_ENABLE(IDC_CALC_START, CanStart),
    EV_COMMAND(IDC_CALC_STOP, Stop),
    EV_COMMAND_ENABLE(IDC_CALC_STOP, CanStop),
    EV_COMMAND(IDC_CALC_AUTO, Automatic),
    EV_COMMAND_ENABLE(IDC_CALC_AUTO, CanStart),
    EV_COMMAND(IDC_HELP_CONTENTS, Help),
    EV_COMMAND(IDC_HELP_ABOUT, About),
END_RESPONSE_TABLE;

// Construct an instance with the specified parent.
TGraphLayoutWindow::TGraphLayoutWindow(TWindow* parent)
: TWindow(parent),
fControl(this, IDD_CONTROLPANEL),
fState(kClosed),
fVertexCount(0),
fEdgeCount(0)
{
}

// Set up the window.
void TGraphLayoutWindow::SetupWindow()
{
    // Perform the default set up.
    TWindow::SetupWindow();

    // Create the control panel.
    fControl.Create();
}

// Paint the window using the specified device context.
void TGraphLayoutWindow::Paint(TDC& dc, bool, TRect&)
{
    // Make sure there is something to draw.
    if (fState == kClosed || fVertexCount == 0)
        return;

    // Set the screen map.
    {
        double xmin = 1e20, xmax = -1e20, ymin = 1e20, ymax
= -1e20;
        double delta;
        for (short i = 0; i < fVertexCount; ++i) {

```

```

        double x, y;
        bool fixed;
        fLayout.GetVertexPosition(i, x, y, fixed);
        xmin = ::min(xmin, x);
        xmax = ::max(xmax, x);
        ymin = ::min(ymin, y);
        ymax = ::max(ymax, y);
    }
    delta = (xmax - xmin) / 20;
    if (delta == 0)
        delta = 1;
    xmin -= delta;
    xmax += delta;
    delta = (ymax - ymin) / 20;
    if (delta == 0)
        delta = 1;
    ymin -= delta;
    ymax += delta;
    fMap.Set(GetClientRect(), xmin, xmax, ymin, ymax);
}

// Draw the edges.
for (short i = 0; i < fEdgeCount; ++i) {
    double x, y;
    bool fixed;
    fLayout.GetVertexPosition(fEdges[i].From(), x, y,
fixed);
    dc.MoveTo(fMap.Unmap(x, y));
    fLayout.GetVertexPosition(fEdges[i].To(), x, y,
fixed);
    dc.LineTo(fMap.Unmap(x, y));
}

// Draw the vertices.
TBrush redBrush(TColor(0xff, 0, 0));
TBrush blueBrush(TColor(0, 0, 0xff));
for (short i = 0; i < fVertexCount; ++i) {
    double x, y;
    bool fixed;
    fLayout.GetVertexPosition(i, x, y, fixed);
    dc.FillRect(TRect(fMap.Unmap(x, y).OffsetBy(-2,
-2), TSize(6, 6)), fixed ? redBrush : blueBrush);
}
}

// Respond to resizing.
void TGraphLayoutWindow::EvSize(uint, TSize&)
{
    // Redraw the display.
    Invalidate();
    UpdateWindow();
}

// Respond to a mouse button going down at the specified
location.
void TGraphLayoutWindow::EvLButtonDown(uint, TPoint&
location)
{
    // Only process this event when in the moving state.
    if (fState == kMoving) {

        // Find the vertex selected.
        FindVertex(location);

        // Enter the editing state if no vertex is under
        the mouse.
        if (fVertex == -1) {
            fState = kEditing;
            Parent->SetCursor(NULL, IDC_ARROW);
            ReleaseCapture();

            // Otherwise, change the cursor to the moving
            cursor.
            } else
                Parent->SetCursor(GetModule(), IDU_MOVE2);
        }
}

// Respond to a mouse button going up at the specified
location.
void TGraphLayoutWindow::EvLButtonUp(uint, TPoint& location)
{
    // If in the fixing state, toggle the fix state of the
    vertex under the mouse, if any.
    if (fState == kFixing) {
        FindVertex(location);
        if (fVertex != -1) {
            double x, y;
            bool fixed;
            fLayout.GetVertexPosition(fVertex, x, y, fixed);
            fLayout.SetVertexPosition(fVertex, x, y,
fixed);
        }
    }

    // If in the moving state, move the selected vertex to
    where the mouse was released.
    } else if (fState == kMoving) {
        double x, y;
        bool fixed;
        fLayout.GetVertexPosition(fVertex, x, y, fixed);
        x = fMap.XMapper().Map(location.X());
        y = fMap.YMapper().Map(location.Y());
        fLayout.SetVertexPosition(fVertex, x, y, fixed);
    }

    // Enter the editing state.
}

```

```

        if (fState == kFixing || fState == kMoving) {
            fState = kEditing;
            Parent->SetCursor(NULL, IDC_ARROW);
            ReleaseCapture();
            Invalidate();
            UpdateWindow();
        }
    }

    // Respond to a request to close the window.
    bool TGraphLayoutWindow::CanClose()
    {
        // Make sure a calculation is not in progress.
        const bool result = fState != kRunning && fState != kStopping;
        if (!result)
            MessageBox("Cannot close while calculating.", "Graph Layout", MB_OK | MB_ICONEXCLAMATION);
        return result;
    }

    // Process the open command.
    void TGraphLayoutWindow::Open()
    {
        // Determine whether to use explorer-style dialog boxes.
        OSVERSIONINFO osVersionInfo;
        osVersionInfo.dwOSVersionInfoSize = sizeof(osVersionInfo);
        GetVersionEx(&osVersionInfo);
        const uint32 explorerFlag = osVersionInfo.dwPlatformId
== VER_PLATFORM_WIN32_WINDOWS || (osVersionInfo.dwPlatformId
== VER_PLATFORM_WIN32_NT && osVersionInfo.dwMajorVersion ==
4) ? OFN_EXPLORER : 0;

        // Get the vertex file name.
        TOpenSaveDialog::TData
vertexFilenameData(OFN_HIDEREADONLY | OFN_FILEMUSTEXIST |
OFN_PATHMUSTEXIST | OFN_EXTENSIONDIFFERENT | OFN_LONGNAMES |
explorerFlag, "Vertex Files (*.ver)|*.ver|All Files (*.*)|*.*|", 0, "", "*");
        if (TFileDialog(this, vertexFilenameData, 0, "Open Vertex File").Execute() != IDOK)
            return;

        // Get the edge file name.
        TOpenSaveDialog::TData edgeFilenameData(OFN_HIDEREADONLY |
OFN_FILEMUSTEXIST | OFN_PATHMUSTEXIST | OFN_EXTENSIONDIFFERENT | OFN_LONGNAMES | explorerFlag, "Edge Files (*.edg)|*.edg|All Files (*.*)|*.*|", 0, "", "*");
        if (TFileDialog(this, edgeFilenameData, 0, "Open Edge File").Execute() != IDOK)
            return;
    }

    // Use the wait cursor.
    Parent->SetCursor(NULL, IDC_WAIT);

    // Count the number of vertices and edges.
    char line[1000];
    fVertexCount = -1;
    fEdgeCount = -1;
    for (ifstream vertexFile(vertexFilenameData.FileName);
!vertexFile.eof(); vertexFile.getline(line, sizeof(line)))
        ++fVertexCount;
    for (ifstream edgeFile(edgeFilenameData.FileName);
!edgeFile.eof(); edgeFile.getline(line, sizeof(line)))
        ++fEdgeCount;
    fLayout.SetVertexCount(fVertexCount);
    fEdges = vector<TEdge>(fEdgeCount);

    // Read the vertices.
    ifstream vertexFile(vertexFilenameData.FileName);
    for (short i = 0; i < fVertexCount; ++i) {
        double x, y;
        short fixed;
        #ifdef STREAM_INPUT_OKAY
            vertexFile >> x >> y >> fixed;
        #else
            vertexFile.getline(line, sizeof(line));
            sscanf(line, "%lf %lf %hi", &x, &y, &fixed);
        #endif
        fLayout.SetVertexPosition(i, x, y, fixed != 0);
    }

    // Read the edges.
    ifstream edgeFile(edgeFilenameData.FileName);
    for (short i = 0; i < fEdgeCount; ++i) {
        short from, to;
        double length;
        #ifdef STREAM_INPUT_OKAY
            edgeFile >> from >> to >> length;
        #else
            edgeFile.getline(line, sizeof(line));
            sscanf(line, "%hi %hi %lf", &from, &to,
&length);
        #endif
        if (length == 0)
            length = 1;
        fEdges[i] = TEdge(from, to);
        fLayout.ConnectVertices(from, to, length);
        if (!fLayout.ConnectVertices(from, to, length)) {
            MessageBox("The vertex and edge\r\nfiles are inconsistent.", "Open", MB_OK | MB_ICONSTOP);
            Close();
            return;
        }
    }
}

```

```

// Set up the graph layout.
fState = kEditing;
fTime = 0;
fFilename = vertexFilenameData.FileName;
Parent->SetCaption(("Graph Layout - " + fFilename).c_str());
Invalidate();
UpdateWindow();
Parent->SetCursor(NULL, IDC_ARROW);
}

// Process the close command.
void TGraphLayoutWindow::Close()
{
    // Clear the graph layout.
    fState = kClosed;
    fVertexCount = 0;
    fEdgeCount = 0;
    Parent->SetCaption("Graph Layout");
    Invalidate();
    UpdateWindow();
}

// Process the save command.
void TGraphLayoutWindow::Save()
{
    // Use the wait cursor.
    Parent->SetCursor(NULL, IDC_WAIT);

    // Save the vertices to the current vertex file.
    ofstream vertexFile(fFilename.c_str());
    for (short i = 0; i < fVertexCount; ++i) {
        double x, y;
        bool fixed;
        fLayout.GetVertexPosition(i, x, y, fixed);
        vertexFile << setprecision(20) << x << "\t" <<
setprecision(20) << y << "\t" << fixed << endl;
    }

    // Return to the arrow cursor.
    Parent->SetCursor(NULL, IDC_ARROW);
}

// Process the save as command.
void TGraphLayoutWindow::SaveAs()
{
    // Determine whether to use explorer-style dialog
    // boxes.
    OSVERSIONINFO osVersionInfo;
    osVersionInfo.dwOSVersionInfoSize = sizeof(osVersionInfo);
    GetVersionEx(&osVersionInfo);
    const uint32 explorerFlag = osVersionInfo.dwPlatformId
== VER_PLATFORM_WIN32_WINDOWS || (osVersionInfo.dwPlatformId
== VER_PLATFORM_WIN32_NT && osVersionInfo.dwMajorVersion == 4) ? OFN_EXPLORER : 0;

    // Get the new vertex file name.
    TOpenSaveDialog::TData filenameData(OFN_HIDEREADONLY |
OFN_PATHMUSTEXIST | OFN_EXTENSIONDIFFERENT |
OFN_OVERWRITEPROMPT | OFN_LONGNAMES | explorerFlag, "Vertex
Files (*.ver)|*.ver|All Files (*.*)|*.*|", 0, "", "*");
    if (TFileDialog(this, filenameData, 0, "Save Vertex
File").Execute() != IDOK)
        return;
    fFilename = filenameData.FileName;
    Parent->SetCaption(("Graph Layout - " + fFilename).c_str());

    // Perform the save operation.
    Save();
}

// Process the fix position command.
void TGraphLayoutWindow::Fix()
{
    // Enter the fixing state.
    fState = kFixing;
    Parent->SetCursor(GetModule(), IDU_FIX);
    SetCapture();
}

// Process the move vertex command.
void TGraphLayoutWindow::Move()
{
    // Enter the moving state.
    fState = kMoving;
    Parent->SetCursor(GetModule(), IDU_MOVE);
    SetCapture();
}

// Process the set lengths command.
void TGraphLayoutWindow::SetLengths()
{
    // Open the set lengths dialog box.
    TEdgeLengthDialog(this, fLayout).Execute(); IDD_EDGELENGTH,
}

// Process the randomize command.
void TGraphLayoutWindow::Randomize()
{
    // Get the randomization parameters by opening the
randomizer dialog box.
    double position, velocity;
    if (TRandomizerDialog(this, IDD_RANDOMIZER, position,
velocity).Execute() == IDOK) {
}

```

```

        // Perform the randomization and update the
display.
        fLayout.RandomizePositions(position);
        fLayout.RandomizeVelocities(velocity);
        Invalidate();
        UpdateWindow();

    }

// Process the paste command.
void TGraphLayoutWindow::Paste()
{
    // Create a metafile with the drawing on it.
    TMetaFileDC dc;
    Paint(dc, false, TRect());
    TMetaFilePict pict(dc.Close(), AutoDelete);

    // Paste the metafile to the clipboard.
    TClipboard clipboard = OpenClipboard();
    clipboard.EmptyClipboard();
    pict.ToClipboard(clipboard,           MM_ISOTROPIC,
GetClientRect().Size());
    clipboard.CloseClipboard();
}

// Process the initialize command.
void TGraphLayoutWindow::Initialize()
{
    // Ask whether to connect all of the vertices.
    const int answer = MessageBox("Put springs
between\r\nall pairs of vertices?", "Initialize",
MB_YESNOCANCEL | MB_ICONQUESTION);
    if (answer == IDCANCEL)
        return;

    // Use the wait cursor.
    Parent->SetCursor(NULL, IDC_WAIT);

    // Initialize the graph layout.
    fLayout.Initialize(answer == IDYES);

    // Update the display.
    Parent->SetCursor(NULL, IDC_ARROW);
    Invalidate();
    UpdateWindow();
}

// Process the start command.
void TGraphLayoutWindow::Start()
{
    // Enter the running state.
    fState = kRunning;
}

```

```

        // Show the control panel.
        fControl.Update();
        fControl.SendDlgItemMessage(IDC_CP_REDRAW, BM_SETCHECK,
1);
        fControl.SendDlgItemMessage(IDC_CP_PAUSED, BM_SETCHECK,
1);
        UpdateControlPanel();
        fControl.ShowWindow(SW_SHOWNORMAL);

        // Start the computation thread.
        DWORD tid;
        CreateThread(NULL, 0, (LPTHREAD_START_ROUTINE)
TGraphLayoutWindow::Compute, this, 0, &tid);
}

// Process the stop command.
void TGraphLayoutWindow::Stop()
{
    // Enter the stopping state.
    fState = kStopping;
    fControl.ShowWindow(SW_HIDE);
}

// Process the automatic command.
void TGraphLayoutWindow::Automatic()
{
    // Use the wait cursor.
    Parent->SetCursor(NULL, IDC_WAIT);

    // Perform the automatic layout and check its results.
    if (!fLayout.LayoutAutomatically())
        MessageBox("Steady state reached before\r\nkinetic
energy damped out.", "Automatic Layout", MB_OK |
MB_ICONEXCLAMATION);

    // Update the display.
    Invalidate();
    UpdateWindow();
    Parent->SetCursor(NULL, IDC_ARROW);
}

// Process the help contents command.
void TGraphLayoutWindow::Help()
{
    // Find the help file.
    char helpFile[1000];
    strcpy(helpFile, _argv[0]);
    strcpy(strrchr(helpFile, '.'), ".hlp");

    // Open the help file.
    WinHelp(helpFile, HELP_INDEX, 0);
}

// Process the help about command.

```

```

void TGraphLayoutWindow::About()
{
    // Display the application information.
    MessageBox("Graph Layout\r\nCopyright © 1996\r\nRegents
of the University of California\r\nAll rights
reserved.\r\n\r\nB. W. Bush\r\nEnergy and Environmental
Analysis Group\r\nnTSA-4, Mail Stop F604\r\nLos Alamos
National Laboratory\r\nLos Alamos, NM 87545\r\nn505-667-6485
/bwb@lanl.gov", "About Graph Layout", MB_OK);
}

// Process an enabling request for opening.
void TGraphLayoutWindow::CanOpen(TCommandEnabler&
commandEnabler)
{
    // The state must be closed or editing.
    commandEnabler.Enable(fState == kClosed || fState ==
kEditing);
}

// Process an enabling request for editing.
void TGraphLayoutWindow::CanEdit(TCommandEnabler&
commandEnabler)
{
    // The state must be editing.
    commandEnabler.Enable(fState == kEditing);
}

// Process an enabling request for starting.
void TGraphLayoutWindow::CanStart(TCommandEnabler&
commandEnabler)
{
    // The state must be editing and the layout must be
initialized.
    commandEnabler.Enable(fState == kEditing &&
fLayout.IsInitialized());
}

// Process an enabling request for stoping.
void TGraphLayoutWindow::CanStop(TCommandEnabler&
commandEnabler)
{
    // The state must be running.
    commandEnabler.Enable(fState == kRunning);
}

// Find the vertex near a point, if any.
void TGraphLayoutWindow::FindVertex(const TPoint& location)
{
    // See which vertex is nearest the point.
    fVertex = -1;
    int nearest = 30000;
    for (short i = 0; i < fVertexCount; ++i) {
        double x, y;
        bool fixed;
        fLayout.GetVertexPosition(i, x, y, fixed);
        const int magnitude = (fMap.Unmap(x, y) -
location).Magnitude();
        if (magnitude < nearest) {
            nearest = magnitude;
            fVertex = i;
        }
    }

    // Make sure the vertex is near enough.
    if (fVertex != -1 && nearest > 3)
        fVertex = -1;
}

// Update the control panel.
void TGraphLayoutWindow::UpdateControlPanel()
{
    // Update the display if necessary.
    if (fIncrement != 0 &&
fControl.SendDlgItemMessage(IDC_CP_REDRAW, BM_GETCHECK) ==
1) {
        Invalidate();
        UpdateWindow();
    }

    // Create streams for input and output.
    char buffer[100];
    istrstream in(buffer);
    ostrstream out(buffer, sizeof(buffer));

    // Update the time and energy in the control panel.
    out.seekp(0, ios::beg);
    out << fTime << ends;
    fControl.SetDlgItemText(IDC_CP_TIME, buffer);
    double kinetic = 0, potential = 0, energy;
    fLayout.GetEnergy(kinetic, potential);
    energy = kinetic + potential;
    out.seekp(0, ios::beg);
    out << kinetic << ends;
    fControl.SetDlgItemText(IDC_CP_KINETIC, buffer);
    out.seekp(0, ios::beg);
    out << potential << ends;
    fControl.SetDlgItemText(IDC_CP_POTENTIAL, buffer);
    out.seekp(0, ios::beg);
    out << energy << ends;
    fControl.SetDlgItemText(IDC_CP_ENERGY, buffer);

    // Update the parameters for the layout.
    fLayout.SetTimeStep(fControl.TimeStep());
    fLayout.SetTemperature(fControl.Temperature());
    fLayout.SetDamping(fControl.Damping());
    fLayout.SetSpringParameters(fControl.A(), fControl.B());
}

```

```

    fIncrement = fControl.SendDlgItemMessage(IDC_CP_PAUSED,
BM_GETCHECK) ? 0 : 1;
}

// Compute the graph layout.
DWORD WINAPI TGraphLayoutWindow::Compute(LPVOID
threadParameter)
{
    // Find the window.
    TGraphLayoutWindow* graphLayoutWindow =
(TGraphLayoutWindow*) threadParameter;

    // Compute as long as the state is running.
    bool warned = false;
    while (graphLayoutWindow->fState == kRunning) {

        // Sleep if the calculation is paused.
        if (graphLayoutWindow->fIncrement == 0)
            Sleep(1000);

        // Otherwise, evolve the vertex positions.
        else {
            graphLayoutWindow-
>fLayout.EvolvePositions(graphLayoutWindow->fIncrement);
            double timeStep;
            graphLayoutWindow-
>fLayout.GetTimeStep(timeStep);
            graphLayoutWindow->fTime += graphLayoutWindow-
>fIncrement * timeStep;
        }

        // Retrieve the energy and make sure the
computation is not diverging.
        double kinetic, potential;
        graphLayoutWindow->fLayout.GetEnergy(kinetic,
potential);
        if (!warned && kinetic > 1e40) {
            if (graphLayoutWindow->MessageBox("The
calculation appears to be diverging.\r\nDo you want to stop
it?", "Calculation", MB_YESNO | MB_ICONEXCLAMATION) ==
IDYES) {
                graphLayoutWindow->fState = kStopping;
                graphLayoutWindow-
>fControl.ShowWindow(SW_HIDE);
            } else
                warned = true;
        }

        // Update the control panel.
        graphLayoutWindow->UpdateControlPanel();
    }

    // Enter the editing state.
}

```

```

graphLayoutWindow->fState = kEditing;
graphLayoutWindow->Invalidate();
graphLayoutWindow->UpdateWindow();
return 0;
}

```

## 6. EdgeLengthDialog.h

```

// Graph Layout Tool
// Copyright © 1996
// Regents of the University of California
// All rights reserved

// B. W. Bush
// Energy and Environmental Analysis Group
// TSA-4, Mail Stop F604
// Los Alamos National Laboratory
// Los Alamos, NM 87545
// 505-667-6485 / bwb@lanl.gov

// $RCSfile: EdgeLengthDialog.h $
// $Revision: 2.0 $
// $Date: 1996/03/06 10:13:35 $

#ifndef LANL_EDGELAYOUTDIALOG
#define LANL_EDGELAYOUTDIALOG

// Include standard C++ library header files.
#include <map>
#include <string>

// Include OWL header files.
#include <owl\pch>
#include <owl\dialog>
#include <owl\listbox>

// Include graph layout tool header files.
#include "GraphLayout.h"

// An edge length dialog allows the user to select how edge
lengths are calculated.
class TEdgeLengthDialog
    : public TDialog
{
public:

    // Construct an instance with the specified parent,
    resource id, and graph layout.
    TEdgeLengthDialog(TWindow* parent, TResId resId,
TGraphLayout& graphLayout);

```

```

// Set up the dialog window.
void SetupWindow();

// Respond to the okay button being pressed.
void CmOk();

// Respond to the help button being pressed.
void CmHelp();

private:

// Use the standard namespace.
using namespace std;

// Type definitions for style maps.
typedef map<std::string, TGraphLayout::EEdgeLengthStyle,
less<std::string> > StyleMap;
typedef StyleMap::iterator StyleMapIterator;

// Each instance has a style map.
StyleMap fStyle;

// Each instance has a style list box.
TListBox fStyleList;

// Each instance references a graph layout.
TGraphLayout& fLayout;

// Declare response table.
DECLARE_RESPONSE_TABLE(TEdgeLengthDialog);
};

#endif // LANL_EDGELAYOUTDIALOG
}

// $RCSfile: EdgeLengthDialog.cpp $
// $Revision: 2.1 $
```

```

// $Date: 1996/03/13 08:42:23 $

// Include standard C++ library header files.
#include <string>
#include <iostream>

// Include OWL header files.
#include <owl\pch>

// Include graph layout tool header files.
#include "EdgeLengthDialog.h"
#include "GraphLayout.rh"

// Define response table.
DEFINE_RESPONSE_TABLE1(TEdgeLengthDialog, TDialog)
    EV_LBN_DBLCLK(IDC_EL_STYLE, CmOk),
    EV_COMMAND(IDOK, CmOk),
    EV_COMMAND(IDHELP, CmHelp),
END_RESPONSE_TABLE;

// Construct an instance with the specified parent,
// resource id, and graph layout.
TEdgeLengthDialog::TEdgeLengthDialog(TWindow* parent, TResId
resId, TGraphLayout& layout)
: TDialog(parent, resId),
fStyleList(this, IDC_EL_STYLE),
fLayout(layout)
{
    // Initialize the style map.
    fStyle["unity"] = TGraphLayout::kUni;
    fStyle["the minimum"] = TGraphLayout::kMin;
    fStyle["the square root of the minimum"] =
TGraphLayout::kSqrtMin;
    fStyle["the maximum"] = TGraphLayout::kMax;
    fStyle["the square root of the maximum"] =
TGraphLayout::kSqrtMax;
    fStyle["the sum"] = TGraphLayout::kSum;
    fStyle["the square root of the sum"] =
TGraphLayout::kSqrtSum;
    fStyle["the square root of the sum of the squares"] =
TGraphLayout::kSqrtSumSqr;
    fStyle["the harmonic sum"] = TGraphLayout::kHarmSum;
    fStyle["the square root of the harmonic sum"] =
TGraphLayout::kSqrtHarmSum;
}

// Set up the dialog window.
void TEdgeLengthDialog::SetupWindow()
{
    // Perform the default set up.
    TDialog::SetupWindow();

    // Fill the style list box.
```

## 7. EdgeLengthDialog.cpp

```

// Graph Layout Tool
// Copyright © 1996
// Regents of the University of California
// All rights reserved

// B. W. Bush
// Energy and Environmental Analysis Group
// TSA-4, Mail Stop F604
// Los Alamos National Laboratory
// Los Alamos, NM 87545
// 505-667-6485 / bwb@lanl.gov

// RCSfile: EdgeLengthDialog.cpp $
// Revision: 2.1 $
```

```

        for (StyleMapIterator i = fStyle.begin(); i != fStyle.end(); ++i)
            fStyleList.AddString((*i).first.c_str());
        fStyleList.SetSelString("the square root of the harmonic
sum", -1);
    }

    // Respond to the okay button being pressed.
void TEdgeLengthDialog::CmOk()
{
    // Get the edge length style and set the edge lengths
accordingly.
    char buffer[100];
    fStyleList.GetSelString(buffer, sizeof(buffer) - 1);
    fLayout.SetEdgeLengths(fStyle[buffer]);

    // Perform the default action.
    TDialog::CmOk();
}

// Respond to the help button being pressed.
void TEdgeLengthDialog::CmHelp()
{
    // Find the help file.
    char helpFile[1000];
    strcpy(helpFile, _argv[0]);
    strcpy(strrchr(helpFile, '.'), ".hlp");

    // Open the help file.
    WinHelp(helpFile, HELP_CONTEXT, IDD_EDGELENGTH);
}

```

## 8. RandomizerDialog.h

```

// Graph Layout Tool
// Copyright © 1996
// Regents of the University of California
// All rights reserved

// B. W. Bush
// Energy and Environmental Analysis Group
// TSA-4, Mail Stop F604
// Los Alamos National Laboratory
// Los Alamos, NM 87545
// 505-667-6485 / bwb@lanl.gov

// $RCSSfile: RandomizerDialog.h $
// $Revision: 2.0 $
// $Date: 1996/03/06 10:13:35 $

#ifndef LANL_RANDOMIZERDIALOG

```

```

#define LANL_RANDOMIZERDIALOG

// Include standard C++ library header files.
#include <string>

// Include OWL header files.
#include <owl\pch>
#include <owl\dialog>
#include <owl\scrollbar>

// A randomizer dialog allows the user to select the
maximum position and velocity perturbation for
randomization.
class TRandomizerDialog
    : public TDialog
{
public:
    // Construct a randomizer dialog with the specified
parent, resource id, position bound, and velocity bound.
    TRandomizerDialog(TWindow* parent, TResId resid, double&
positionBound, double& velocityBound);

    // Set up the dialog window.
    void SetupWindow();

    // Update the numeric values in the dialog.
    void Update();

    // Return the position value.
    double Position() const;

    // Return the velocity value.
    double Velocity() const;

    // Respond to the okay button being pressed.
    void CmOk();

    // Respond to the help button being pressed.
    void CmHelp();

private:
    // Each instance references a maximum position for
randomization.
    double& fPositionBound;

    // Each instance references a maximum velocity for
randomization.
    double& fVelocityBound;

    // Each instance has a position scroll bar.
    TScrollBar fPosition;

```

```

// Each instance has a velocity scroll bar.
TScrollBar fVelocity;

// Declare the response table.
DECLARE_RESPONSE_TABLE(TRandomizerDialog);
};

#endif // LANL_RANDOMIZERDIALOG

```

## 9. RandomizerDialog.cpp

```

// Graph Layout Tool
// Copyright © 1996
// Regents of the University of California
// All rights reserved

// B. W. Bush
// Energy and Environmental Analysis Group
// TSA-4, Mail Stop F604
// Los Alamos National Laboratory
// Los Alamos, NM 87545
// 505-667-6485 / bwb@lanl.gov

// RCSfile: RandomizerDialog.cpp $
// $Revision: 2.0 $
// $Date: 1996/03/06 10:13:35 $

// Include standard C++ library header files.
#include <string>
#include <strstream>

// Include OWL header files.
#include <owl\pch>

// Include graph layout tool header files.
#include "RandomizerDialog.h"
#include "GraphLayout.rh"

// Define response table.
DEFINE_RESPONSE_TABLE1(TRandomizerDialog, TDIALOG)
    EV_SB_ENDSCROLL(IDC_RA_POSITIONSB, Update),
    EV_SB_ENDSCROLL(IDC_RA_VELOCITYSB, Update),
    EV_COMMAND(IDOK, CmOk),
    EV_COMMAND(IDHELP, CmHelp),
END_RESPONSE_TABLE;

// Construct an instance with the specified parent,
resource id, position bound, and velocity bound.

```

```

TRandomizerDialog::TRandomizerDialog(TWindow* parent, TResId
resId, double& positionBound, double& velocityBound)
: TDIALOG(parent, resId),
fPositionBound(positionBound),
fVelocityBound(velocityBound),
fPosition(this, IDC_RA_POSITIONSB),
fVelocity(this, IDC_RA_VELOCITYSB)
{
}

// Set up the dialog window.
void TRandomizerDialog::SetupWindow()
{
    // Perform the default set up.
    TDIALOG::SetupWindow();

    // Set up the position scroll bar.
    fPosition.SetRange(0, 509);
    fPosition.SetPageMagnitude(10);
    fPosition.SetLineMagnitude(1);
    fPosition.SetPosition(100);

    // Set up the velocity scroll bar.
    fVelocity.SetRange(0, 509);
    fVelocity.SetPageMagnitude(10);
    fVelocity.SetLineMagnitude(1);
    fVelocity.SetPosition(100);

    // Update the numeric values in the dialog.
    Update();
}

// Update the numeric values in the dialog.
void TRandomizerDialog::Update()
{
    // Construct an output stream.
    char buffer[100];
    ostrstream out(buffer, sizeof(buffer));

    // Update the position value.
    out.seekp(0, ios::beg);
    out << Position() << ends;
    SetDlgItemText(IDC_RA_POSITION, buffer);

    // Update the velocity value.
    out.seekp(0, ios::beg);
    out << Velocity() << ends;
    SetDlgItemText(IDC_RA_VELOCITY, buffer);
}

// Return the position value.
double TRandomizerDialog::Position() const
{
    // Return the position value.

```

```

        return fPosition.GetPosition() / 100.;

    }

    // Return the velocity value.
double TRandomizerDialog::Velocity() const
{
    // Return the velocity value.
    return fVelocity.GetPosition() / 100.;

}

// Respond to the OK button being pressed.
void TRandomizerDialog::CmOk()
{
    // Set the position and velocity bounds.
fPositionBound = Position();
fVelocityBound = Velocity();

    // Perform the default action.
TDialog::CmOk();
}

// Respond to the help button being pressed.
void TRandomizerDialog::CmHelp()
{
    // Find the help file.
char helpFile[1000];
strcpy(helpFile, _argv[0]);
strcpy(strrchr(helpFile, '.'), ".hlp");

    // Open the help file.
WinHelp(helpFile, HELP_CONTEXT, IDD_RANDOMIZER);
}

```

## 10. ControlPanelDialog.h

```

// Graph Layout Tool
// Copyright © 1996
// Regents of the University of California
// All rights reserved

// B. W. Bush
// Energy and Environmental Analysis Group
// TSA-4, Mail Stop F604
// Los Alamos National Laboratory
// Los Alamos, NM 87545
// 505-667-6485 / bwb@lanl.gov

// $RCSfile: ControlPanelDialog.h $
// $Revision: 2.0 $
// $Date: 1996/03/06 10:13:35 $

```

```

#ifndef LANL_CONTROLPANELDIALOG
#define LANL_CONTROLPANELDIALOG

// Include standard C++ library header files.
#include <string>

// Include OWL header files.
#include <owl\pch>
#include <owl\dialog>
#include <owl\scrollbar>

// A control panel dialog allows the user to adjust the
graph layout calculation interactively.
class TControlPanelDialog
    : public TDialog
{
public:
    // Construct an instance with the specified parent and
resource id.
    TControlPanelDialog(TWindow* parent, TResId resId);

    // Set up the dialog window.
    void SetupWindow();

    // Update the numeric values in the dialog.
    void Update();

    // Return the time step value.
    double TimeStep() const;

    // Return the temperature value.
    double Temperature() const;

    // Return the damping value.
    double Damping() const;

    // Return the A parameter value.
    double A() const;

    // Return the B parameter value.
    double B() const;

    // Respond to the help button being pressed.
    void CmHelp();

private:
    // Each instance has a time step scroll bar.
    TScrollBar fTimeStep;

    // Each instance has a temperature scroll bar.
    TScrollBar fTemperature;

```

```

// Each instance has a damping scroll bar.
TScrollBar fDamping;

// Each instance has a A parameter scroll bar.
TScrollBar fA;

// Each instance has a B parameter scroll bar.
TScrollBar fB;

// Declare the response table.
DECLARE_RESPONSE_TABLE(TControlPanelDialog);
};

#endif // LANL_CONTROLPANELDIALOG

// Graph Layout Tool
// Copyright © 1996
// Regents of the University of California
// All rights reserved

// B. W. Bush
// Energy and Environmental Analysis Group
// TSA-4, Mail Stop F604
// Los Alamos National Laboratory
// Los Alamos, NM 87545
// 505-667-6485 / bwb@lanl.gov

// RCSfile: ControlPanelDialog.cpp $
// $Revision: 2.0 $
// $Date: 1996/03/06 10:13:35 $

// Include standard C++ library header files.
#include <string>
#include <iostream>

// Include OWL header files.
#include <owl\pch>

// Include graph layout tool header files.
#include "ControlPanelDialog.h"
#include "GraphLayout.rh"

// Define response table.
DEFINE_RESPONSE_TABLE1(TControlPanelDialog, TDIALOG)
    EV_SB_ENDSCROLL(IDC_CP_TIMESTEPSB, Update),
    EV_SB_ENDSCROLL(IDC_CP_TEMPERATURESB, Update),
    EV_SB_ENDSCROLL(IDC_CP_DAMPINGSB, Update),
    EV_SB_ENDSCROLL(IDC_CP_ASB, Update),
    EV_SB_ENDSCROLL(IDC_CP_BSB, Update),
    EV_COMMAND(IDHELP, CmHelp),
END_RESPONSE_TABLE;

// Construct an instance with the specified parent and
resource id.
TControlPanelDialog::TControlPanelDialog(TWindow* parent,
TResId resId)
: TDIALOG(parent, resId),
fTimeStep(this, IDC_CP_TIMESTEPSB),
fTemperature(this, IDC_CP_TEMPERATURESB),
fDamping(this, IDC_CP_DAMPINGSB),
fA(this, IDC_CP_ASB),
fB(this, IDC_CP_BSB)
{
}

// Set up the dialog window.
void TControlPanelDialog::SetupWindow()
{
    // Perform the default set up.
    TDIALOG::SetupWindow();

    // Set up the time step scroll bar.
    fTimeStep.SetRange(1, 1099);
    fTimeStep.SetPageMagnitude(100);
    fTimeStep.SetLineMagnitude(10);
    fTimeStep.SetPosition(100);

    // Set up the temperature scroll bar.
    fTemperature.SetRange(0, 2099);
    fTemperature.SetPageMagnitude(100);
    fTemperature.SetLineMagnitude(10);
    fTemperature.SetPosition(0);

    // Set up the damping factor scroll bar.
    fDamping.SetRange(0, 2099);
    fDamping.SetPageMagnitude(100);
    fDamping.SetLineMagnitude(10);
    fDamping.SetPosition(1000);

    // Set up the spring parameters scroll bars.
    fA.SetRange(0, 3099);
    fA.SetPageMagnitude(100);
    fA.SetLineMagnitude(10);
    fA.SetPosition(0);
    fB.SetRange(0, 6099);
    fB.SetPageMagnitude(100);
    fB.SetLineMagnitude(10);
    fB.SetPosition(4000);

    // Update the numeric values in the dialog.
    Update();
}

```

```

}

// Update the numeric values in the dialog.
void TControlPanelDialog::Update()
{
    // Construct an output stream.
    char buffer[100];
    ostrstream out(buffer, sizeof(buffer));

    // Update the time step value.
    out.seekp(0, ios::beg);
    out << TimeStep() << ends;
    SetDlgItemText(IDC_CP_TIMESTEP, buffer);

    // Update the temperature value.
    out.seekp(0, ios::beg);
    out << Temperature() << ends;
    SetDlgItemText(IDC_CP_TEMPERATURE, buffer);

    // Update the damping factor value.
    out.seekp(0, ios::beg);
    out << Damping() << ends;
    SetDlgItemText(IDC_CP_DAMPING, buffer);

    // Update the spring parameters values.
    out.seekp(0, ios::beg);
    out << A() << ends;
    SetDlgItemText(IDC_CP_A, buffer);
    out.seekp(0, ios::beg);
    out << B() << ends;
    SetDlgItemText(IDC_CP_B, buffer);
}

// Return the time step value.
double TControlPanelDialog::TimeStep() const
{
    // Return the time step value.
    return fTimeStep.GetPosition() / 1000.0;
}

// Return the temperature value.
double TControlPanelDialog::Temperature() const
{
    // Return the temperature value.
    return fTemperature.GetPosition() / 1000.0;
}

// Return the damping value.
double TControlPanelDialog::Damping() const
{
    // Return the damping value.
    return fDamping.GetPosition() / 1000.0;
}

// Return the A parameter value.
double TControlPanelDialog::A() const
{
    // Return the A parameter value.
    return fA.GetPosition() / 1000.0;
}

// Return the B parameter value.
double TControlPanelDialog::B() const
{
    // Return the B parameter value.
    return (fB.GetPosition() - 3000) / 1000.0;
}

// Respond to the help button being pressed.
void TControlPanelDialog::CmHelp()
{
    // Find the help file.
    char helpFile[1000];
    strcpy(helpFile, _argv[0]);
    strcpy(strrchr(helpFile, '.'), ".hlp");

    // Open the help file.
    WinHelp(helpFile, HELP_CONTEXT, IDD_CONTROLPANEL);
}

```

## 12. Mapper.h

```

// Graph Layout Tool
// Copyright © 1996
// Regents of the University of California
// All rights reserved

// B. W. Bush
// Energy and Environmental Analysis Group
// TSA-4, Mail Stop F604
// Los Alamos National Laboratory
// Los Alamos, NM 87545
// 505-667-6485 / bwb@lanl.gov

#ifndef LANL_MAPPER
#define LANL_MAPPER

// Rectangle template class.
template <class ATYPE>
struct TRectangle
{
    // Each instance has a left boundary.
    ATYPE fLeft;

```

```

// Each instance has a top boundary.
AType fTop;

// Each instance has a right boundary.
AType fRight;

// each instance has a bottom boundary.
AType fBottom;
};

// Rectangle equality operator.
template <class AType>
int operator==(const TRectangle<AType>& a, const
TRectangle<AType>& b)
{
    return a.fLeft == b.fLeft && a.fRight == b.fRight &&
a.fTop == a.fTop && a.fBottom == a.fBottom;
}

// Rectangle inequality operator.
template <class AType>
int operator!=(const TRectangle<AType>& a, const
TRectangle<AType>& b)
{
    return a.fLeft != b.fLeft || a.fRight != b.fRight || a.fTop != a.fTop || a.fBottom != a.fBottom;
}

// Mapper template class.
template <class ATypeX, class ATypeY>
class TMapper
{
public:

    // Construct an instance.
    TMapper()
        : fTruncate(false),
        fLogarithmic(false) {}

    // Set the mapping from the domain (x0, x1) to the
    // range (y0, y1), making the map truncate values and/or
    // logarithmic if indicated.
    void Set(ATypeX x0, ATypeX x1, ATypeY y0, ATypeY y1,
bool truncate = false, bool logarithmic = false)
    {
        fX0 = x0;
        fX1 = x1;
        fY0 = y0;
        fY1 = y1;
        fTruncate = truncate;
        fLogarithmic = logarithmic;
    }

    // Map the specified value to the range.
    ATypeY Map(const ATypeX x) const
    {
        if (fTruncate && x < fX0 && x < fX1)
            return fX0 < fX1 ? fY0 : fY1;
        else if (fTruncate && x > fX1 && x > fX0)
            return fX1 > fX0 ? fY1 : fY0;
        else
            return fLogarithmic ? (ATypeY) expl(fA * x + fB)
: (ATypeY) (fA * x + fB);
    }

    // Map the specified value to the domain.
    ATypeX Unmap(const ATypeY y) const
    {
        if (fTruncate && y < fY0 && y < fY1)
            return fY0 < fY1 ? fX0 : fX1;
        else if (fTruncate && y > fY1 && y > fY0)
            return fY1 > fY0 ? fX1 : fX0;
        else
            return fLogarithmic ? (ATypeX) ((logl((long
double) y) - fB) / fA) : (ATypeX) ((y - fB) / fA);
    }

    // Map the specified difference to the range.
    ATypeY MapDifference(const ATypeX x = 1) const
    {
        return fLogarithmic ? (ATypeY) expl(fA * x) :
(ATypeY) (fA * x);
    }

    // Map the specified difference to the domain.
    ATypeX UnmapDifference(const ATypeY y = 1) const
    {
        return fLogarithmic ? (ATypeX) (logl((long double)
y) / fA) : (ATypeX) (y / fA);
    }

    // Return whether the specified value is in the domain.
    bool InsideDomain(const ATypeX x) const
    {
}

```

```

        return fx0 < fx1 ? fx0 <= x && x <= fx1 : fx1 <= x
&& x <= fx0;
}

// Return whether the specified value is in the domain.
bool InsideRange(const ATypeY y) const
{
    return fy0 < fy1 ? fy0 <= y && y <= fy1 : fy1 <= y
&& y <= fy0;
}

// Return the left side of the domain.
ATypeX DomainLeft() const
{
    return fx0;
}

// Return the right side of the domain.
ATypeX DomainRight() const
{
    return fx1;
}

// Return the left side of the range.
ATypeY RangeLeft() const
{
    return fy0;
}

// Return the right side of the range.
ATypeY RangeRight() const
{
    return fy1;
}

private:
    // Each instance has a slope parameter.
    long double fA;

    // Each instance has an intercept parameter.
    long double fB;

    // Each instance has a flag indicating whether the map
is logarithmic.
    bool fLogarithmic;

    // Each instance has a flag indicating whether the map
should truncate results.
    bool fTruncate;

    // Each instance has a left side to its domain.
    ATypeX fx0;
}

// Each instance has a right side to its domain.
ATypeX fx1;

// Each instance has a left side to its range.
ATypeY fy0;

// Each instance has a right side to its range.
ATypeY fy1;
};

// ScreenMapper template class.
template <class ATypeX, class ATypeY>
class TScreenMapper
{
public:
    // Construct an instance.
    TScreenMapper()
        : fY(), fX() {}

    // Set the map to be from the rectangle r to the
rectangle (x0, x1) x (y0, y1), making the map truncate
values and/or logarithmic if indicated.
    void Set(const TRect &r, ATypeX x0, ATypeX x1, ATypeY
y0, ATypeY y1, bool truncate = false, bool xLog = false,
bool yLog = false)
    {
        fX.Set(r.left, r.right, x0, x1, truncate, xLog);
        fY.Set(r.bottom, r.top, y0, y1, truncate, yLog);
    }

    // Map the point (x, y) to the domain.
    TPoint Unmap(ATypeX x, ATypeY y) const
    {
        return TPoint(fX.Unmap(x), fY.Unmap(y));
    }

    // Return the abscissa map.
    TMapper<int, ATypeX>& XMapper()
    {
        return fX;
    }
    const TMapper<int, ATypeX>& XMapper() const
    {
        return fX;
    }

    // Return the ordinate map.
    TMapper<int, ATypeY>& YMapper()
    {
        return fY;
    }
    const TMapper<int, ATypeY>& YMapper() const

```

```

{
    return fy;
}

private:

    // Each instance has an abscissa map.
    TMapper<int, ATYPEX> fx;

    // Each instance has an ordinate map.
    TMapper<int, ATYPEY> fy;
};

#endif // LANL_MAPPER

```

```

#define IDD_RANDOMIZER      1302
#define IDD_CONTROLPANEL   1303

    // Define accelerator resource ids.
#define IDA_GRAPHAYOUT     1401

    // Define command ids for file menu.
#define IDC_FILE_OPEN       2101
#define IDC_FILE_CLOSE      2102
#define IDC_FILE_SAVE        2103
#define IDC_FILE_SAVEAS     2104
#define IDC_FILE_EXIT        CM_EXIT

    // Define command ids for edit menu.
#define IDC_EDIT_FIX         2201
#define IDC_EDIT_MOVE        2202
#define IDC_EDIT_LENGTH      2203
#define IDC_EDIT_RANDOMIZE   2204
#define IDC_EDIT_PASTE        2205

    // Define command ids for calculation menu.
#define IDC_CALC_INITIALIZE 2301
#define IDC_CALC_START        2302
#define IDC_CALC_STOP         2303
#define IDC_CALC_AUTO         2304

    // Define command ids for help menu.
#define IDC_HELP_CONTENTS    2401
#define IDC_HELP_ABOUT       2402

    // Define command ids for edge length dialog box.
#define IDC_EL_STYLE         3101

    // Define command ids for randomizer dialog box.
#define IDC_RA_POSITION      3201
#define IDC_RA_POSITIONSB    3202
#define IDC_RA_VELOCITY      3203
#define IDC_RA_VELOCITYSB    3204

    // Define command ids for control panel dialog.
#define IDC_CP_TIMESTEP      3301
#define IDC_CP_TIMESTEPSB    3302
#define IDC_CP_TEMPERATURE    3303
#define IDC_CP_TEMPERATURESB  3304
#define IDC_CP_DAMPING        3305
#define IDC_CP_DAMPINGSB     3306
#define IDC_CP_A               3307
#define IDC_CP_ASB             3308
#define IDC_CP_B               3309
#define IDC_CP_BSB             3310
#define IDC_CP_TIME            3311
#define IDC_CP_KINETIC         3312
#define IDC_CP_POTENTIAL       3313
#define IDC_CP_ENERGY          3314

```

### 13. GraphLayout.rh

```

// Graph Layout Tool
// Copyright © 1996
// Regents of the University of California
// All rights reserved

// B. W. Bush
// Energy and Environmental Analysis Group
// TSA-4, Mail Stop F604
// Los Alamos National Laboratory
// Los Alamos, NM 87545
// 505-667-6485 / bwb@lanl.gov

// $RCSfile: GraphLayout.rh $
// $Revision: 2.1 $
// $Date: 1996/03/13 08:37:44 $

#ifndef LANL_GRAPHAYOUT_RH
#define LANL_GRAPHAYOUT_RH

// Define icon resource ids.
#define IDI_GRAPHAYOUT      1001

// Define cursor resource ids.
#define IDU_FIX              1101
#define IDU_MOVE             1102
#define IDU_MOVE2            1103

// Define menu resource ids.
#define IDM_GRAPHAYOUT      1201

// Define dialog resource ids.
#define IDD_EDGELength      1301

```

```

#define IDC_CP_REDRAW      3315
#define IDC_CP_PAUSED      3316

// Define bitmap resource ids.
#define IDC_FILE_OPEN       4000 + IDC_FILE_OPEN
#define IDC_FILE_SAVE       4000 + IDC_FILE_SAVE
#define IDC_EDIT_FIX        4000 + IDC_EDIT_FIX
#define IDC_EDIT_MOVE        4000 + IDC_EDIT_MOVE
#define IDC_EDIT_LENGTH      4000 + IDC_EDIT_LENGTH
#define IDC_EDIT_RANDOMIZE   4000 + IDC_EDIT_RANDOMIZE
#define IDC_CALC_INITIALIZE  4000 + IDC_CALC_INITIALIZE
#define IDC_CALC_START        4000 + IDC_CALC_START
#define IDC_CALC_STOP         4000 + IDC_CALC_STOP
#define IDC_CALC_AUTO         4000 + IDC_CALC_AUTO
#define IDC_HELP_CONTENTS    4000 + IDC_HELP_CONTENTS
#define IDC_HELP_ABOUT        4000 + IDC_HELP_ABOUT

#endif // LANL_GRAPH_LAYOUT_RH

14. GraphLayout.rc
// Graph Layout Tool
// Copyright © 1996
// Regents of the University of California
// All rights reserved

// B. W. Bush
// Energy and Environmental Analysis Group
// TSA-4, Mail Stop F604
// Los Alamos National Laboratory
// Los Alamos, NM 87545
// 505-667-6485 / bwb@lanl.gov

// $RCSfile: GraphLayout.rc $
// $Revision: 2.3 $
// $Date: 1996/09/18 14:28:16 $

// Include OWL header files.
#include <owl\window.rh>

// Include graph layout tool header files.
#include "GraphLayout.rh"

// Main menu.
IDM_GRAPH_LAYOUT MENU
{
    POPUP "&File"
    {
        MENUITEM "&Open...", IDC_FILE_OPEN
        MENUITEM "&Close", IDC_FILE_CLOSE
    }
}

// Edge length dialog box.
IDD_EDGELENGTH DIALOG 6, 15, 194, 115
STYLE DS_MODALFRAME | WS_POPUP | WS_VISIBLE | WS_CAPTION | WS_SYSMENU
CAPTION "Set Edge Lengths"
FONT 8, "MS Sans Serif"
{
    CONTROL "How to combine the count of edges at each of an edge's endpoints into the length for the edge:", -1,
    "static", SS_LEFT | WS_CHILD | WS_VISIBLE, 8, 4, 172, 22
    CONTROL "", IDC_EL_STYLE, "listbox", LBS_STANDARD | WS_CHILD | WS_VISIBLE | WS_TABSTOP, 8, 30, 180, 65
    DEFPUSHBUTTON "OK", IDOK, 12, 95, 50, 14
    PUSHBUTTON "Cancel", IDCANCEL, 72, 95, 50, 14
    PUSHBUTTON "Help", IDHELP, 132, 95, 50, 14
}

// Randomizer dialog box.
IDD_RANDOMIZER DIALOG 6, 15, 194, 63
STYLE DS_MODALFRAME | WS_POPUP | WS_VISIBLE | WS_CAPTION | WS_SYSMENU
CAPTION "Randomize Vertices"
FONT 8, "MS Sans Serif"
{
    MENUITEM SEPARATOR
    MENUITEM "&Save", IDC_FILE_SAVE
    MENUITEM "Save &as...", IDC_FILE_SAVEAS
    MENUITEM SEPARATOR
    MENUITEM "E&xit", IDC_FILE_EXIT
}
POPUP "&Edit"
{
    MENUITEM "&Fix position\af", IDC_EDIT_FIX
    MENUITEM "&Move vertex\am", IDC_EDIT_MOVE
    MENUITEM "&Set lengths...\al", IDC_EDIT_LENGTH
    MENUITEM "&Randomize...\ar", IDC_EDIT_RANDOMIZE
    MENUITEM SEPARATOR
    MENUITEM "&Paste", IDC_EDIT_PASTE
}
POPUP "&Calculation"
{
    MENUITEM "&Initialize...", IDC_CALC_INITIALIZE
    MENUITEM "St&art", IDC_CALC_START
    MENUITEM "St&op", IDC_CALC_STOP
    MENUITEM SEPARATOR
    MENUITEM "Automatic", IDC_CALC_AUTO
}
POPUP "&Help"
{
    MENUITEM "&Contents", IDC_HELP_CONTENTS
    MENUITEM "&About", IDC_HELP_ABOUT
}
}

LA-UR-96-2166

```

```

LTEXT "Position:", -1, 5, 9, 30, 8
LTEXT "", IDC_RA_POSITION, 40, 10, 30, 8
SCROLLBAR IDC_RA_POSITIONSB, 75, 10, 110, 9, SBS_HORZ |
WS_TABSTOP
LTEXT "Velocity:", -1, 5, 25, 30, 8
LTEXT "", IDC_RA_VELOCITY, 40, 25, 30, 8
SCROLLBAR IDC_RA_VELOCITYSB, 75, 25, 110, 9, SBS_HORZ |
WS_TABSTOP
DEFPUSHBUTTON "OK", IDOK, 12, 43, 50, 14
PUSHBUTTON "Cancel", IDCANCEL, 72, 43, 50, 14
PUSHBUTTON "Help", IDHELP, 132, 43, 50, 14
}

// Control panel dialog box.
IDD_CONTROLPANEL DIALOG 9, 19, 234, 149
STYLE WS_POPUP | WS_CAPTION | WS_MINIMIZEBOX
CAPTION "Calculation Control"
FONT 8, "MS Sans Serif"
{
    LTEXT "Time step:", -1, 10, 10, 60, 8
    LTEXT "", IDC_CP_TIMESTEP, 70, 10, 50, 10
    SCROLLBAR IDC_CP_TIMESTEPSB, 120, 10, 105, 9, SBS_HORZ |
    WS_TABSTOP
    LTEXT "Temperature:", -1, 10, 25, 60, 12
    LTEXT "", IDC_CP_TEMPERATURE, 70, 25, 50, 10
    SCROLLBAR IDC_CP_TEMPERATURESB, 120, 25, 105, 9,
    SBS_HORZ | WS_TABSTOP
    LTEXT "Damping:", -1, 10, 40, 60, 8
    LTEXT "", IDC_CP_DAMPING, 70, 40, 50, 10
    SCROLLBAR IDC_CP_DAMPINGSB, 120, 40, 105, 9, SBS_HORZ |
    WS_TABSTOP
    LTEXT "Parameter A:", -1, 10, 65, 60, 12
    LTEXT "", IDC_CP_A, 70, 65, 50, 10
    SCROLLBAR IDC_CP_ASB, 120, 65, 105, 9, SBS_HORZ |
    WS_TABSTOP
    LTEXT "Parameter B:", -1, 10, 80, 60, 8
    LTEXT "", IDC_CP_B, 70, 80, 50, 10
    SCROLLBAR IDC_CP_BSB, 120, 80, 105, 9, SBS_HORZ |
    WS_TABSTOP
    LTEXT "Kinetic energy:", -1, 10, 105, 60, 8
    LTEXT "", IDC_CP_KINETIC, 70, 105, 50, 10
    LTEXT "Potential energy", -1, 10, 120, 60, 8
    LTEXT "", IDC_CP_POTENTIAL, 70, 120, 50, 10
    LTEXT "Total energy:", -1, 10, 135, 60, 8
    LTEXT "", IDC_CP_ENERGY, 70, 135, 50, 10
    LTEXT "Time:", -1, 130, 105, 45, 8
    LTEXT "", IDC_CP_TIME, 175, 105, 50, 10
    AUTOCHECKBOX "Redraw", IDC_CP_REDRAW, 130, 120, 40, 12
    AUTOCHECKBOX "Pause", IDC_CP_PAUSED, 130, 135, 40, 12
    PUSHBUTTON "Help", IDHELP, 175, 125, 50, 14
}

// String table for hints.
STRINGTABLE
{
    IDC_FILE_OPEN,
    file."
    IDC_FILE_CLOSE,
    IDC_FILE_SAVE,
    IDC_FILE_SAVEAS,
    different name."
    IDC_FILE_EXIT,
    IDC_EDIT_FIX,
    IDC_EDIT_MOVE,
    IDC_EDIT_LENGTH,
    to a particular style."
    IDC_EDIT_RANDOMIZE,
    velocities."
    IDC_EDIT_PASTE,
    clipboard."
    IDC_CALC_INITIALIZE,
    IDC_CALC_START,
    calculation."
    IDC_CALC_STOP,
    calculation."
    IDC_CALC_AUTO,
    automatically."
    IDC_HELP_CONTENTS,
    IDC_HELP_ABOUT,
    information."
}

// Command accelerators.
IDA_GRAPHAYOUT ACCELERATORS
{
    "f", IDC_EDIT_FIX
    "m", IDC_EDIT_MOVE
    "l", IDC_EDIT_LENGTH
    "r", IDC_EDIT_RANDOMIZE
}

// Main icon.
IDI_GRAPHAYOUT ICON
{
    '00 00 01 00 01 00 20 20 10 00 00 00 00 00 E8 02'
    '00 00 16 00 00 00 28 00 00 00 20 00 00 00 40 00'
    '00 00 01 00 04 00 00 00 00 00 80 02 00 00 00 00'
    '00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00'
    '00 00 00 00 80 00 00 80 00 00 00 80 80 00 80 00'
    '00 00 80 00 80 00 80 80 00 00 80 80 80 00 C0 C0'
    'C0 00 00 00 FF 00 00 FF 00 00 00 FF FF 00 00 FF 00'
    '00 00 FF 00 FF 00 FF FF 00 00 FF FF FF 00 00 00'
    '00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00'
    '00 00 00 00 00 00 00 00 00 00 00 00 99 99 90 00'
    '00 00 00 00 00 00 00 09 99 99 99 99 00 00 90 00'
    '00 00 00 00 09 99 90 00 90 00 00 09 00 00 09 00'
    '00 00 00 00 09 00 00 00 90 00 00 00 90 00 00 00'
    '00 00 00 00 09 00 00 00 90 00 00 00 90 00 00 00'
}

```





```

'88 88 88 08 80 88 88 88 00 00 88 88 88 88 88 88 80'
'80 88 88 88 00 00 88 88 88 88 88 88 88 00 88 88 88'
'00 00 88 88 88 88 88 88 00 00 88 88 88 88 00 00 88 88'
'88 88 88 88 88 88 88 88 00 00 88 88 88 88 88 88 88 88'
'88 88 08 88 00 00 88 88 88 88 88 88 88 88 80 00 88'
'00 00 88 88 88 88 88 88 88 08 88 00 00 88 88'
'88 88 88 88 88 88 88 88 00 00 88 88 88 88 88 88 88 88'
'88 88 88 88 00 00'
}

// Set lengths command bitmap.
IDB_EDIT_LENGTH BITMAP
{
  '42 4D 66 01 00 00 00 00 00 00 00 00 76 00 00 00 28 00'
  '00 00 14 00 00 00 14 00 00 00 01 00 04 00 00 00 00'
  '00 00 F0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00'
  '00 00 00 00 00 00 00 00 00 00 00 00 00 00 80 00 00 80'
  '00 00 00 80 80 00 80 00 00 80 00 80 00 80 00 80 00 80 80'
  '00 00 80 80 80 00 C0 C0 00 00 00 FF 00 00 FF'
  '00 00 00 FF FF 00 FF 00 00 00 FF 00 FF 00 FF FF'
  '00 00 FF FF FF 00 88 88 88 88 88 88 88 88 88 88 88 88'
  '00 00 88 80 00 88 88 88 88 88 88 88 88 88 88 88 88 00'
  'E0 08 88 88 88 88 88 88 00 00 80 0E EE 00 08 88 88 88 88'
  '88 88 88 88 00 00 80 EE E0 00 08 88 88 88 88 88 88 88'
  '00 00 80 0E E0 00 88 88 88 88 88 00 00 88 00'
  'EE EE E0 08 88 88 88 88 00 00 88 80 0E EE 00 00'
  '88 88 88 88 00 00 88 88 00 EE 00 E0 08 88 88 88 88'
  '00 00 88 88 80 0E EE 00 88 88 88 00 00 88 88'
  '88 00 EE E0 00 08 88 88 00 00 88 88 88 80 0E E0'
  'OE 00 88 88 00 00 88 88 88 88 00 EE EE E0 08 88'
  '00 00 88 88 88 88 80 0E EE 00 00 88 00 00 88 88'
  '88 88 88 00 EE 00 E0 08 00 00 88 88 88 88 88 88 80'
  'OE EE EE 08 00 00 88 88 88 88 88 00 EE E0 08'
  '00 00 88 88 88 88 80 0E 00 00 88 00 00 88 88'
  '88 88 88 88 00 08 88 00 00 88 88 88 88 88 88 88'
  '88 88 88 88 00 00 88 88 00 00 88 88 88 88 88 88'
  }

// Randomize command bitmap.
IDB_EDIT_RANDOMIZE BITMAP
{
  '42 4D 66 01 00 00 00 00 00 00 00 00 76 00 00 00 28 00'
  '00 00 14 00 00 00 14 00 00 00 01 00 04 00 00 00 00'
  '00 00 F0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00'
  '00 00 00 00 00 00 00 00 00 00 00 00 00 00 80 00 00 80'
  '00 00 00 80 80 00 80 00 00 80 00 80 00 80 00 80 00 80 80'
  '00 00 80 80 80 00 C0 C0 00 00 00 FF 00 00 FF'
  '00 00 00 FF FF 00 FF 00 00 00 FF 00 FF 00 FF FF'
  '00 00 FF FF FF 00 88 88 88 88 88 88 88 88 88 88 88'
  '00 00 88 88 88 88 88 88 00 00 88 80 08 88 88 88 88 88'
  '08 80 08 88 88 88 88 88 00 00 88 80 08 88 88 88 88 88'
  '00 00 88 88 88 88 88 88 00 00 88 80 08 88 88 88 88 88'
  '88 88 88 88 88 88 88 88 00 00 88 80 08 88 88 88 88 88'
  '88 88 88 88 88 88 88 88 00 00 88 80 08 88 88 88 88 88'
  }

// Start command bitmap.
IDB_CALC_START BITMAP
{
  '42 4D 66 01 00 00 00 00 00 00 00 00 76 00 00 00 28 00'
  '00 00 14 00 00 00 14 00 00 00 01 00 04 00 00 00 00'
  '00 00 F0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00'
  '00 00 00 00 00 00 00 00 00 00 00 00 00 00 80 00 00 80'
  '00 00 00 80 80 00 80 00 00 80 00 80 00 80 00 80 00 80 80'
  '00 00 80 80 80 00 C0 C0 00 00 00 FF 00 00 FF'
  '00 00 FF FF FF 00 88 88 88 88 88 88 88 88 88 88 88 88'
  '00 00 88 88 88 88 88 88 00 00 88 80 08 88 88 88 88 88'
  '88 80 00 00 08 88 88 88 00 00 88 88 88 88 88 88 88 88'
  '00 88 88 88 88 88 88 88 00 00 88 80 AA AA A0 00 88 88'
}

```

```

'00 00 88 88 00 AA AA AA AA 00 88 88 00 00 88 80'
'0A AA AA AA AA A0 08 88 00 00 88 00 AA AA AA AA'
'AA AA 00 88 00 00 88 00 AA AA AA AA AA 00 88'
'00 00 88 00 AA AA AA AA AA 00 88 00 00 88 00'
'AA AA AA AA AA 00 88 00 00 88 00 AA AA AA AA AA'
'AA AA 00 88 00 00 88 00 AA AA AA AA AA 00 88'
'00 00 88 80 0A AA AA AA A0 08 88 00 00 88 88'
'00 AA AA AA AA 00 88 88 00 00 88 88 00 0A AA AA'
'A0 00 88 88 00 00 88 88 00 00 00 00 88 88 88'
'00 00 88 88 88 80 00 00 08 88 88 88 00 00 88 88'
'88 88 88 88 88 88 88 00 00 88 88 88 88 88 88 88'
'88 88 88 88 00 00'
}

// Stop command bitmap.
IDB_CALC_STOP_BITMAP
{
  '42 4D 66 01 00 00 00 00 00 00 76 00 00 00 28 00'
  '00 00 14 00 00 00 14 00 00 00 01 00 04 00 00 00'
  '00 00 F0 00 00 00 00 00 00 00 00 00 00 00 00 00 00'
  '00 00 00 00 00 00 00 00 00 00 00 00 80 00 00 80'
  '00 00 00 80 80 00 80 00 00 80 00 00 80 00 00 80 80'
  '00 00 80 80 80 00 C0 C0 C0 00 00 00 FF 00 00 FF'
  '00 00 00 FF FF 00 FF 00 00 00 FF 00 FF 00 FF FF'
  '00 00 FF FF FF 00 88 88 88 88 88 88 88 88 88 88'
  '00 00 88 88 88 88 88 88 88 88 88 88 00 00 88 88'
  '88 80 00 00 08 88 88 88 00 00 88 88 88 00 00 00 00'
  '00 88 88 88 00 00 88 88 00 09 99 99 99 90 00 88 88'
  '00 00 88 88 00 99 99 99 99 00 88 88 00 00 88 80'
  '09 99 99 99 99 90 08 88 00 00 88 00 99 99 99 99'
  '99 99 00 88 00 00 88 00 99 99 99 99 99 99 99 00 88'
  '00 00 88 00 99 99 99 99 99 00 88 00 00 88 00'
  '99 99 99 99 99 99 00 88 00 00 88 00 99 99 99 99'
  '99 99 00 88 00 00 88 00 99 99 99 99 99 99 99 00 88'
  '00 00 88 00 99 99 99 99 99 00 88 00 00 88 00'
  '00 00 88 80 09 99 99 99 99 90 08 88 00 00 88 88'
  '00 99 99 99 99 00 88 88 00 00 88 88 00 09 99 99'
  '90 00 88 88 00 00 88 88 88 00 00 00 88 88 88'
  '00 00 88 88 88 80 00 00 08 88 88 88 00 00 88 88'
  '88 88 88 88 88 88 88 88 00 00 88 88 88 88 88 88'
  '88 88 88 88 00 00'
}

// Automatic command bitmap.
IDB_CALC_AUTO_BITMAP
{
  '42 4D 66 01 00 00 00 00 00 00 76 00 00 00 28 00'
  '00 00 14 00 00 00 14 00 00 00 01 00 04 00 00 00'
  '00 00 F0 00 00 00 00 00 00 00 00 00 00 00 00 00 00'
  '00 00 00 00 00 00 00 00 00 00 00 00 80 00 00 80'
  '00 00 80 80 00 80 00 00 00 80 00 80 00 80 80 80'
  '00 00 80 80 80 00 C0 C0 C0 00 00 00 FF 00 00 FF'
  '00 00 00 FF FF 00 FF 00 00 00 FF 00 FF 00 FF FF'
  '00 00 FF FF FF 00 88 88 88 88 88 88 88 88 88 88'
  '2A 32 88 88 88 88 88 88 88 88 88 88 8C 3E 88 88'
}

// Help contents command bitmap.
IDB_HELP_CONTENTS_BITMAP
{
  '42 4D 66 01 00 00 00 00 00 00 76 00 00 00 28 00'
  '00 00 14 00 00 00 14 00 00 00 01 00 04 00 00 00'
  '00 00 F0 00 00 00 00 00 00 00 00 00 00 00 00 00 00'
  '00 00 00 00 00 00 00 00 00 00 00 00 80 00 00 80'
  '00 00 80 80 80 00 C0 C0 C0 00 00 00 FF 00 00 FF'
  '00 00 00 FF FF 00 FF 00 00 00 FF 00 FF 00 FF FF'
  '00 00 FF FF FF 00 88 88 88 88 88 88 88 88 88 88 88'
  '55 90 80 44 44 44 44 40 88 88 88 88 88 88 0B 6A 80 06'
  '66 06 66 00 88 88 88 88 11 D4 80 F0 00 00 00 60'
  '88 88 88 88 44 72 80 8F E6 0F FE 70 88 88 88 88'
  'B3 28 80 F8 86 0F 88 60 88 88 88 88 88 CC 3C 80 8F'
  'E6 0F FE 70 88 88 88 88 2A B8 80 F8 86 0F 88 60'
  '88 88 88 88 04 34 84 0F E6 0F FE 04 88 88 88 88'
  '22 39 88 80 00 80 00 88 88 00 08 88 22 E2 88 88'
  '88 88 88 88 88 0E 08 88 9C 80 88 88 88 88 88 88'
  '88 00 08 88 28 BA 88 88 88 88 88 88 88 88 0E 08 88'
  '00 00 88 88 88 88 88 88 88 88 88 88 0E 00 88 00 00 88 88'
  '88 88 88 88 00 00 E0 08 00 00 88 88 88 88 88 88 88 88'
  '0E 00 0E 08 00 00 88 88 88 88 88 88 88 0E 00 0E 08'
  '00 00 88 88 88 88 88 88 88 88 88 88 00 EE E0 08 00 00 88 88'
  '88 88 88 88 80 00 00 88 00 00 88 88 88 88 88 88 88 88'
  '88 88 88 88 00 00'
}

// Help about command bitmap.
IDB_HELP_ABOUT_BITMAP
{
  '42 4D 66 01 00 00 00 00 00 00 76 00 00 00 28 00'
  '00 00 14 00 00 00 14 00 00 00 01 00 04 00 00 00'
  '00 00 F0 00 00 00 00 00 00 00 00 00 00 00 00 00 00'
  '00 00 00 00 00 00 00 00 00 00 00 00 80 00 00 80'
  '00 00 80 80 00 80 00 00 00 80 00 80 00 80 80 80'
  '00 00 80 80 80 00 C0 C0 C0 00 00 00 FF 00 00 FF'
  '00 00 00 FF FF 00 FF 00 00 00 FF 00 FF 00 FF FF'

```

```

'00 00 FF FF FF 00 88 88 88 88 88 88 88 88 88 88 88 88'
'46 00 88 88 88 80 00 00 08 88 88 88 00 00 88 88'
'80 04 00 44 40 08 88 88 11 02 88 88 04 40 FF 04'
'44 40 88 88 08 00 88 80 44 0F FF F0 44 44 08 88'
'46 00 88 04 44 0F F0 78 04 44 40 88 00 00 88 04'
'44 47 FF 00 44 44 40 88 33 00 80 44 44 40 FF 74'
'44 44 44 08 08 00 80 44 44 44 7F F0 44 44 44 08'
'46 00 80 44 44 40 0F F7 44 44 44 08 00 00 80 44'
'44 08 70 FF 04 44 44 08 11 02 80 44 44 40 FF FF'
'04 44 44 08 5B 17 80 44 44 44 0F F0 44 44 44 08'
'CE 88 88 04 44 44 40 04 44 44 40 88 00 00 88 04'
'44 44 44 00 44 44 40 88 11 02 88 80 44 44 40 FF'
'04 44 08 88 4C 67 88 88 04 44 40 FF 04 40 88 88'
'46 08 88 88 80 04 44 00 40 08 88 88 00 00 88 88'
'88 80 00 00 08 88 88 88 13 00 88 88 88 88 88 88'
'88 88 88 88 7F 63'
}

// Version information.
IDV_GRAPH_LAYOUT VERSIONINFO
FILEVERSION 1, 0, 0, 0
PRODUCTVERSION 1, 0, 0, 0
FILEOS VOS_NT_WINDOWS32
FILETYPE VFT_APP
{
    BLOCK "StringFileInfo"
    {
        BLOCK "040904E4"
        {
            VALUE "CompanyName",     "Los Alamos National
Laboratory\000\000"
            VALUE "FileDescription", "Graph Layout Tool\000"
            VALUE "FileVersion",   "Revision: 2.3 \000\000"
            VALUE "InternalName",  "GraphLayoutWin\000"
            VALUE "LegalCopyright", "Copyright © 1996
Regents of the University of California\000\000"
            VALUE "OriginalFilename",
"GraphLayoutWin.exe\000"
        }
    }
    BLOCK "VarFileInfo"
    {
        VALUE "Translation", 0x409, 1252
    }
}

```